TRƯỜNG ĐẠI HỌC ĐÀ LẠT

GIÁO TRÌNH

KỸ THUẬT LẬP TRÌNH NÂNG CAO

TRẦN HOÀNG THỌ

2002

MỤC LỤC

LỜI NÓI ĐẦU

Giáo trình được viết theo nội dung môn học " Kỹ thuật lập trình nâng cao" với mục đích làm tài liệu tham khảo chính cho môn học.

Giáo trình gồm 2 phần chính và một phụ lục :

Phần I. Đệ quy.

Trình bày về chủ đề đệ quy trong lập trình bao gồm các nội dung sau :

- Khái niệm đệ quy và vai trò của nó trong lập trình.
- Cách xây dựng một giải thuật cho một bài toán bằng phương pháp đệ quy.
- Cơ chế thực hiện một giải thuật đệ quy.
- Khử đệ quy.

Phần II. Kiểm chứng chương trình.

Trình bày về chủ đề kiểm chứng tính đúng của chương trình bao gồm các nội dung sau:

- Vai trò của vấn đề kiểm chứng trong lập trình.
- Các phương pháp dùng để kiểm chứng tính đúng .
- Hệ luật Hoare và áp dụng của nó vào kiểm chứng tính đúng có điều kiện.
- Hệ luật Dijkstra và áp dụng của nó vào kiểm chứng tính đúng đầy đủ.
- Dạng tổng quát của bài toán kiểm chứng và phương pháp kiểm chứng. Các lược đồ kiểm chứng và tập tối thiểu các điều kiện cần kiểm chứng.

Phụ lục . Các kiến thức chung về logic.

Trình bày các kiến thức ban đầu về logic mệnh đề và logic tân từ. Phụ lục cung cấp một một tài liệu cô đọng về các kiến thức logic áp dụng trực tiếp trong phần I và phần II (nó là một phần nôi dung của giáo trình nhập môn toán) người học cần dành thời gian thích hợp ôn lại để có thể theo kịp hướng tiếp cận của giáo trình.

Cùng với những trình bày lý thuyết tổng quát, tác gỉa đưa vào một số thỏa đáng các ví dụ chọn lọc nhằm giúp người học nắm bắt được bản chất của các khái niệm, các phương pháp mới và làm quen với cách sử dụng các kết qủa mới. Khi học trước khi tìm cách giải các bài tập của thầy gíao cung cấp các bạn cố gắng đọc và hiểu hết các ví dụ minh họa.

Vì nhiều lẽ chắc chắn giáo trình còn nhiều khiếm khuyết. Rất mong tất cả mọi người sử dụng chân thành góp ý.

Tác giả chân thành cảm ơn các đồng nghiệp trong khoa Toán_Tin đã đóng góp nhiều ý kiến quý báu cho việc hình thành cấu trúc chi tiết cho nội dung giáo trình, chân thành cảm ơn thạc sỹ Võ Tiến đã đóng góp nhiều ý kiến quý báu trong cấu trúc giáo trình, giúp chỉnh lý nhiều khiếm khuyết trong bản thảo.

ĐaLat ngày 01 tháng 12 năm 2002

TRẦN HOÀNG THỌ

PHẦN I

ĐỆ QUY

CHƯƠNG I
KHÁI NIỆM ĐỆ QUY

I. MỞ ĐẦU

1. Mô tả đệ quy

Trong nhiều tình huống việc mô tả các bài toán, các giải thuật, các sự kiện, các sự vật các quá trình, các cấu trúc, . . . sẽ đơn giản và hiệu quả hơn nếu ta nhìn được nó dưới góc độ mang tính đệ qui.

Mô tả mang tính đệ qui về một đối tượng là mô tả theo cách phân tích đối tượng thành nhiều thành phần mà trong số các thành phần có thành phần mang tính chất của chính đối tượng được mô tả. Tức là mô tả đối tượng qua chính nó.

Các ví dụ :

- Mô tả đệ quy tập số tự nhiên N :
    + Số 1 là số tự nhiên ($1 \in N$) .
    + Số tự nhiên bằng số tự nhiên cộng 1 .
      ($n \in N \Rightarrow (n+1) \in N$)
- Mô tả đệ quy cấu trúc xâu (list) kiểu T :
    + Cấu trúc rỗng là một xâu kiểu T.
    + Ghép nối một thành phần kiểu T(nút kiểu T) với một xâu kiểu T ta có một xâu kiểu T.
- Mô tả đệ quy cây gia phả : Gia phả của một người bao gồm mgười đó và gia phả của cha và gia phả của mẹ.
- Mô tả đê quy thủ tục chọn hoa hậu :
    + Chọn hoa hậu của từng khu vực.
    + Chọn hoa hậu của các hoa hậu.
- Mô tả đệ quy thủ tục sắp tăng dãy a[m:n] (dãy a[m], a[m+1], . . . , a[n]) bằng phương pháp Sort_Merge (SM) :

SM (a[m:n]) ≡ Merge (SM(a[m : (n+m) div 2]) , SM (a[(n+m) div 2 +1 : n])

Với : SM (a[x : x]) là thao tác rỗng (không làm gì cả).

Merge (a[x : y] , a[(y+1) : z]) là thủ tục trộn 2 dãy tăng a [x : y] , a[(y+1) : z] để được một dãy a[x : z] tăng.

- Đinh nghĩa đệ quy hàm giai thừa FAC(n) = n !

$0 ! = 1$

$n ! = n * (n - 1) !$

Phương pháp đệ quy mạnh ở chổ nó cho phép mô tả một tập lớn các đối tượng chỉ bởi một số ít các mệnh đề hoặc mô tả một giải thuật phức tạp bằng một số ít các thao tác (một chương trình con đệ quy).

Một mô tả đệ quy đầy đủ gồm 2 phần :

- Phần neo : mô tả các trường hợp suy biến của đối tượng (giải thuật) qua một cấu trúc (thao tác) cụ thể xác định .

ví dụ: 1 là số tự nhiên, cấu trúc rỗng là một xâu kiểu T, 0 ! = 1 , SM (a[x:x]) là thao tác rỗng.

- Phần quy nạp: mô tả đối tượng (giải thuật) trong trường hợp phổ biến thông qua chính đối tượng (giải thuật) đó một cách trực tiếp hoặc gián tiếp.

Ví dụ : n! = n * (n – 1) !

SM (a[m:n]) ≡ Merge (SM (a[m:(m+n) div 2] , SM (a[(m+n) div 2 +1 : n]))

Nếu trong mô tả không có phần neo thì đối tượng mô tả có cấu trúc lớn vô hạn, giải thuật mô tả trở thành cấu trúc lặp vô tận.

2. Các loại đệ quy

Người ta phân đệ quy thành 2 loại : Đệ quy trực tiếp, đệ quy gián tiếp.

- Đệ quy trực tiếp là loại đệ quy mà đối tượng được mô tả trực tiếp qua nó : A mô tả qua A, B, C,...trong đó B, C, ... không chứa A. (các ví dụ trên).

- Đệ quy gián tiếp là loại đệ quy mà đối tượng được mô tả gián tiếp qua nó : A mô tả qua A_1 ,A_2 ,..., A_n .Trong đó có một A_i được mô tả qua A.

Ví dụ 1:

Mô tả dạng tổng quát một chương trình viết trên NNLT Pascal :

Một Chương trình Pascal gồm :

a) Đầu chương trình (head) gồm: Program Tên **;**

b) Thân chương trình (blok) gồm :

b1) Khai báo unit, định nghĩa hằng, nhãn, kiểu dữ liệu, khái báo biến.

b2) Định nghĩa các chương trình con gồm :

b2.1) Đầu chương trình con :

Procedure Tên thủ tục (danh sách thông số hình thức) **;**

hoặc Function Tên hàm (danh sách thông số hình thức) **:** Kiểu **;**

b2.2) Thân chương trình con (Blok)

b2.3) Dấu **‘ ; ‘**

b3) Phần lệnh : là một lệnh ghép dạng :

Begin S1 ; S2 ; . . . ; Sn End ;

c) Dấu kết thúc chương trình : **‘.’**

Ví dụ 2 : Mô tả hai dãy số $\{X_n\},\{Y_n\}$ theo luật đệ quy hổ tương như sau :

$X_0 = 1$; $X_n = X_{n-1} + Y_{n-1}$;

$Y_0 = 1$; $Y_n = n^2 X_{n-1} + Y_{n-1}$;

II. MÔ TẢ ĐỆ QUY CÁC CẤU TRÚC DỮ LIỆU

Trong toán học , trong lập trình người ta thường sử dụng đệ quy để mô tả các cấu trúc phức tạp, có tính đệ quy . Bởi mô tả đệ quy không chỉ là cách mô tả ngắn gọn các cấu trúc phức tạp mà còn tạo khả năng để xây dựng các thao tác xử lý trên các cấu trúc phức tạp bằng các giải thuật đệ qui . Một cấu trúc dữ liệu có tính đệ quy thường gồm một số thành phần dữ liệu cùng kiểu được ghép nối theo cùng một phương thức .

Ví dụ 1:

Mô tả đệ quy cây nhi phân :

Cây nhi phân kiểu T :

+ Hoặc là một cấu trúc rỗng (phần neo).

+ Hoặc là một nút kiểu T (nút gốc) và 2 cây nhị phân kiểu T rời nhau (cây con nhị phân phải, cây con nhị phân trái) kết hợp với nhau .

Ví dụ 2:

Mô tả đệ quy mảng nhiều chiều :

+ Mảng một chiều là dãy có thứ tự các thành phần cùng kiểu .

+ Mảng n chiều là mảng 1 chiều mà các thành phần có kiểu mảng n-1 chiều .

III. MÔ TẢ ĐỆ QUY GIẢI THUẬT

1. Giải thuật đệ quy.

Giải thuật đệ quy là giải thuật có chứa thao tác gọi đến nó . Giải thuật đệ quy cho phép mô tả một dãy lớn các thao tác bằng một số ít các thao tác trong đó có chứa thao tác gọi lại giải thuật (gọi đệ quy) .

Một cách tổng quát một giải thuật đệ quy được biểu diễn như một bộ P gồm mệnh đề S (không chứa yếu tố đệ quy) và P : $P \equiv P[S, P]$.

Thực thi giải thuật đệ quy có thể dẫn tới một tiến trình gọi đê quy không kết thúc khi nó không có khả năng gặp trường hợp neo, vì vậy quan tâm đến điều kiện dừng của một giải thuật đệ quy luôn được đặt ra . Để kiểm soát qúa trình gọi đệ quy của giải thuật đệ quy P người ta thường gắn thao tác gọi P với việc kiểm tra một điều kiện B xác định và biến đổi qua mỗi lần gọi P , qúa trình gọi P sẻ dừng khi B không con thỏa.

Mô hình tổng quát của một giải thuật đệ quy với sự quan tâm đến sự dừng sẻ là :

P ≡ if B then P[S , P]

hoặc P ≡ P[S , if B then P]

Thông thường với giải thuật đệ quy P , để đảm bảo P sẻ dừng sau n lần gọi ta chọn B là (n >0) . Mô hình giải thuật đệ quy khi đó có dạng :

P(n) ≡ If (n > 0) then P[S , P(n - 1)] ;

hoặc P(n) ≡ P[S , if (n >0) then P(n - 1)] ;

Trong các ứng dụng thực tế số lần gọi đệ quy (độ sâu đệ quy) không những phải hữu hạn mà còn phải đủ nhỏ . Bởi vì mỗi lần gọi đệ quy sẽ cần một vùng nhớ mới trong khi vùng nhớ cũ vẫn phải duy trì .

2. Chương trình con đệ quy.

a) Các hàm đệ quy.

Định nghĩa hàm số bằng đệ quy thường gặp trong toán học, điển hình là các hàm nguyên mô tả các dãy số hồi quy .

Ví dụ 1 .

Dãy các giai thừa : { n! } ≡ 1 ,1 , 2 , 6 , 24 , 120 , 720 , 5040 , . . .

Ký hiệu FAC(n) = n ! .

Ta có : + FAC(0) = 1 ; (0 ! = 1)

+ FAC(n) = n * FAC(n - 1) ; (n ! = n * (n - 1) !) với n >= 1

Giải thuật đệ quy tính FAC(n) là :

```
FAC(n ) ≡ if (n = 0 ) then return 1 ;
                      else return (n * FAC(n - 1 )) ;
```

Ví dụ 2 .

Dãy số Fibonaci(FIBO) :

{ FIBO (n) } ≡ 1 ,1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , . . .

+ FIBO(0) = FIBO (1) = 1 ;

+ FIBO(n) = FIBO (n - 1) + FIBO (n - 2) ; với n > = 2

Giải thuật đệ quy tính FIBO (n) là :

```
FIBO(n) ≡ if ((n = 0 ) or ( n = 1 )) then return 1 ;
                      else return ( FIBO (n - 1) + FIBO (n - 2)) ;
```

Ví dụ 3 . Dãy các tổ hợp :

1

1 2 1

1 3 3 1

1 4 6 4 1

$C_n^0 = 1$ với n > = 0

$C_n^m = 0$ với m > n > 0

$C_n^m = C_{n-1}^{m-1} + C_{n-1}^m$ với n > m > 0

Giải thuật đệ quy tính C_n^m là :

if (m = 0) then return 1 ;
else if (m > n) then return 0 ;
else return ($C_{n-1}^{m-1} + C_{n-1}^m$) ;

Nhận xét :

Một định nghĩa hàm đệ quy gồm :

+ Một số các trường hợp suy biến mà gía trị hàm tại đó đã được biết trước hoặc có thể tính một cách đơn giản (không đệ quy) .

Như :

FAC(0) = 1 , FIBO(0) = FIBO(1) = 1 , C_n^0 = 1 , C_n^m = 0 với m > n > 0 .

+ Trường hợp tổng quát việc tính hàm sẻ đươc đưa về tính hàm ở giá trị " bé hơn" (gần với giá trị neo) của đối số .

Như :

FAC(n) = n * FAC(n - 1) ;

FIBO(n) = FIBO(n -1) + FIBO(n - 2) .

Trong tập biến của hàm có một nhóm mà độ lớn của nó quyết định độ phức tạp của việc tính gía trị hàm . Nhóm biến đó gọi là nhóm biến điều khiển . Gía trị biên của nhóm biến điều khiển ứng với trường hợp suy biến . Gía trị của nhóm biến điều khiển sẻ thay đổi qua mỗi lần gọi đệ quy với xu hướng tiến đến gía trị biên (tương ứng với các trường hợp suy biến của hàm).

b) Các thủ tục đệ quy.

Thủ tục đệ quy là thủ tục có chứa lệnh gọi đến nó . Thủ tục đệ quy thường được sử dụng để mô tả các thao tác trên cấu trúc dữ liệu có tính đệ quy

Ví dụ 1 :

Xem dãy n phần tử a[1:n] là sự kết hợp giữa dãy a[1:n-1] và a[n] .

Do đó :

- Thủ tục tìm max trong dãy a[1:n] (thủ tục TMax) có thể thực hiện theo luật đệ qui : + Tìm max trong dãy con a[1:n] (gọi đệ quy Tmax(a[1:n-1])).

+ Tìm max của 2 số : Tmax(a[1:n-1]) và a[n] (giải thuật không đệ quy).

Tức là :

TMax(a[1:n]) = max(TMax(a[1:n-l]) , a[n])

với TMax(a[m:m] = a[m] ; (trường hợp neo)

max(x,y) = x > y ? x : y ; (giải thuật tính max 2 số : if (x>y) then max(x ,y) = x else max(x ,y) = y)

- Thủ tục tính tổng các phần tử (thủ tục TSUM) có thể thực hiện theo luật đệ quy :

+ Tìm tổng dãy con a[1:n] (gọi đệ quy TSUM(a[1:n-1])).

+ Tìm tổng của 2 số : TSUM(a[1:n-1]) và a[n] (giải thuật không đệ quy).

Tức là :

TSUM(a[1:n]) = a[n] + TSUM(a[1:n-1]

với TSUM(a[m:m]) = a[m]

Ví dụ 2 :

Xem dãy a[m : n] là sự kết nối giữa hai dãy: dãy a[m:((m+n) div 2)] và dãy a[(((m+n) div 2)+1) :n] .

Do đó :

- Thủ tục tìm max trong dãy a[1:n] (thủ tục Tmax1) có thể thực hiện theo luật đệ qui :
  + Tìm max trong dãy con trái a[m:((m+n) div 2)]
    (gọi đệ quy Tmax1(a[m:((m+n) div 2)])).
  + Tìm max trong dãy con phải a[(((m+n) div 2)+1) :n] .
    (gọi đệ quy Tmax1(a[(((m+n) div 2)+1) :n]).
  + Tìm max của 2 số : Tmax1(a[m:((m+n) div 2)]) và
    Tmax1(a[(((m+n) div 2)+1) :n]). (giải thuật không đệ quy).

Tức là :Tmax1(a[m:n]) =
max(Tmax1(a[m:((m+n) div 2)]) ,Tmax1(a[(((m+n) div 2)+1) :n])).
với Tmax1(a[m:m] = a[m] ; (trường hợp neo)
max(x,y) = x > y ? x : y ;

- Thủ tục tính tổng các phần tử (TSUM1) có thể thực hiện theo luật đệ quy :
  + Tìm tổng dãy con trái a[m:((m+n) div 2)]
    (gọi đệ quy TSUM1 (a[m:((m+n) div 2)])).
  + Tìm tổng dãy con phải a[(((m+n) div 2)+1) :n] .
    (gọi đệ quy TSUM1 (a[(((m+n) div 2)+1) :n])).
  + Tìm tổng của 2 số :
    TSUM1 (a[m:((m+n) div 2)]) và TSUM1 (a[(((m+n) div 2)+1) :n]).

Tức là : TSUM1 (a[m:n]) =
TSUM1 (a[m:((m+n) div 2)]) + TSUM1 (a[(((m+n) div 2)+1) :n])
với TSUM1 (a[m:m]) = a[m]

Ví dụ 3 :

Cây nhị phân tìm kiếm kiểu T(BST) là một cấu trúc gồm : một nút kiểu T kết nối với 2 cây con nhi phân tìm kiếm kiểu T nên :

- Thụ tục quét cây nhi nhân tìm kiếm theo thứ tự giữa (LNF) là :
  + Quét cây con trái theo thứ tự giữa ;
  + Thăm nút gốc ;
  + Quét cây con phải theo thứ tự giữa ;
- Thủ tục tìm kiếm giá tri α_o trên cây nhị phân tìm kiếm Root là :

Nếu Root $\equiv \varnothing$ thì thực hiện thao tác rỗng (không làm gì)
Con không
  nếu giá trị tại nút gốc = α_o thì thông báo tìm thấy và dừng
  Còn không
    nếu giá trị tại nút gốc < α_o thì tìm ở cây con trái
      Còn không thì tìm ở cây con phải .

Nhận xét :

Trong một thủ tục đệ qui, để cho việc gọi đệ quy dừng lại sau hữu hạn lần gọi nó cần chứa điều kiện kiểm tra (một biểu thức boolean B trên một nhóm biến) , để khi điều kiện này không còn thỏa thì việc gọi đệ qui kết thúc .

Dạng thường gặp của thủ tục đệ qui là :

S_1 ; (không chứa yếu tố đệ qui)
if B then S_2 (phần lệnh trực tiếp , không có lệnh gọi đệ qui)
else Sdq ; (phần lệnh có lệnh gọi đệ qui)
S3 ; (không có gọi đệ qui)

3. Mã hóa giải thuật đệ qui trong các ngôn ngữ lập trình.

a) Tổng quan.

Không phải mọi ngôn ngữ lập trình hiện có đều có thể mã hóa được giải thuật đệ quy, chỉ một số những ngôn ngữ lập trình có khả năng tổ chức vùng nhớ kiểu stack mới có khả năng mã hóa được giải thuật đệ quy .

Các ngôn ngữ lập trình hiện nay đều mã hóa giải thuật đệ quy bằng cách tổ chức các chương trình con đệ quy tương ứng .

b) Thể hiện đệ qui trong NNLT PASCAL và C++

NN LT Pascal và C++ đều cho phép mã hóa giải thuật đệ quy bằng cách tổ chức chương trình con đê quy nhờ vào cơ chế tạo vùng nhớ Stak của phần mềm ngôn ngữ .

b1) Trong NNLT C++.

NNLT C++ cho phép mã hóa giải thuật đệ quy một cách thuận lợi nhờ vào kỹ thuật khai báo trước tiêu đề nên không có sự phân biệt hình thức nào trong việc khai báo giữa hàm con đệ quy và hàm con không đệ quy.

b2) Trong NN LT Pascal .

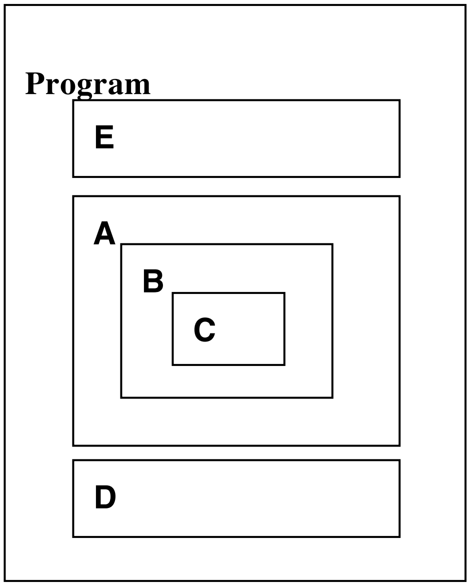
Đối với chương trình con đệ quy trực tiếp thì hình thức khai báo cũng giống như đối với chương trình con không đệ quy.

Đối với chương trình con đệ quy gián tiếp thì hình thức khai báo có thay đổi ít nhiều nhằm thỏa quy tắc tầm vực của ngôn ngữ (trong phần lệnh của một chương trình con chỉ được gọi những chương trình con cùng cấp đã được khai báo trước).

Ví dụ :

Với mô hình chương trình sau :

Trong phần lệnh của khối A có thể gọi đến :



+ Gọi các chương trình con trực tiếp của nó gọi được B nhưng không gọi được C
+ Gọi chính nó (gọi đệ quy).
+ Gọi chương trình con cùng cấp nhưng phải khai báo trước gọi được E nhưng không gọi được D , Muốn gọi D phải khai báo trước (khai báo FORWARD)

Để từ thủ tục hàm A có thể gọi đến D là thủ tục hàm cùng cấp nhưng được mô tả sau A, ta cần có một khai báo trước của D ở phía trước của A . Khai báo này gồm : tiêu đề của D, với danh sách thông số của D, tiếp theo là từ khoá FORWARD . Sau đó lúc mô tả lại D thì chỉ cần khai báo từ khoá PROCEDURE (hoặc FUNCTION) , tên của D (không có danh sách thông số) , phần thân của D.

Ví dụ : Với 2 thủ tục gọi đệ quy hỗ tương nhau FIRST,SECOND sẽ được khai báo như sau :

```
procedure  SECOND (i : integer ) ; Forward ;
procedure  FIRST (n : integer ; var X : real);
            var  j, k : interger ;
             begin
                 for  j :=  1  to  n  do  begin
                                          writeln(‘ j = ‘, j ) ;
                                           k  := n – 2* j  ;
                                          SECOND( k );
                                        end  ;
              end ;
             procedure  second ;
                  begin
                      if ( i > 0 )  then   begin
                                            writeln(‘ i= ‘, i );
                                            FIRST( i – 1 )  ;
                                          end ;

                   end ;
```

4. Một số dạng giải thuật đệ quy đơn giản thường gặp .

a) Đệ quy tuyến tính.

Chương trình con đệ quy tuyến tính là chương trình con đệ quy trực tiếp đơn giản nhất có dạng :

P ≡ { **NẾU** thỏa điều kiện dừng **thì** thực hiện S ;
Còn không begin { thực hiện S* ; gọi P }
}

Với S , S* là các thao tác không đệ quy .

Ví dụ 1 : Hàm FAC(n) tính số hạng n của dãy n!

+ Dạng hàm trong ngôn ngữ mã giả :

{ **Nếu** n = 0 **thì** FAC = 1 ; /* trường hợp neo */
Còn không FAC = n*FAC(n-1) }

+ Dạng hàm trong ngôn ngữ Pascal :

```
Function FAC(n : integer) : integer;
  begin
     if( n = 0 ) then FAC := 1
                 else FAC := n*FAC(n-1) ;
  end;
```

+ Dạng hàm trong C++ :

```
int FAC( int n )
{ if ( n == 0 ) return 1 ;
  else return ( n * FAC(n-1 )) ;
 }
```

Ví dụ 2 :

Chương trình con tính USCLN của 2 số dựa vào thuật toán Euclide :

+ Dạng hàm trên ngôn ngữ toán học :

USCLN(m , n) = USCLN(n , m mod n) khi n ≠ 0
USCLN(m , 0) = m

+ Dạng hàm trong ngôn ngữ mã giả :

Nếu n = 0 **thì** USCLN = m
Còn không USCLN = USCLN(n , m mod n) ;

+ Dạng hàm trong Pascal :

```
Function USCLN(m , n : integer ) : integer ;
  begin
     if (n = 0 ) then USCLN := m
                 else USCLN := USCLN( n , m mod n ) ;
  end ;
```

+Dạng hàm trong C++ :

```
int USCLN( int m , int n )
```

```
{ if(n == 0 ) return (m) ;
  else return ( USCLN( n , m mod n)) ;
  }
```

b) Đệ quy nhị phân.

Chương trình con đệ quy nhị phân là chương trình con đệ quy trực tiếp có dạng :

P ≡ { **NẾU** thỏa điều kiện dừng **thì** thực hiện S ;
Còn không begin { thực hiện S* ; gọi P ; gọi P }
}

Với S , S* là các thao tác không đệ quy .

Ví dụ 1 : Hàm FIBO(n) tính số hạng n của dãy FIBONACCI

+ Dạng hàm trong Pascal:

```
Function F(n : integer) : integer;
  begin
    if( n < 2 ) then F := 1
                else F := F(n-1) + F(n-2)
  end;
```

+ Dạng hàm trong C++ :

```
int F(int n)
  { if ( n < 2 ) return 1 ;
    else return (F(n -1) + F(n -2)) ;
}
```

c) Đệ quy phi tuyến.

Chương trình con đệ quy phi tuyến là chương trình con đệ quy trực tiếp mà lời gọi đệ quy được thực hiện bên trong vòng lặp .

Dạng tổng quát của chương trình con đệ quy phi tuyến là :

P ≡ { **for** giá tri đầu **to** giá trị cuối **do**
begin **thực hiện S ;**
if (thỏa điều kiện dừng) then thực hiện S*
else gọi P
end ;
}

Với S , S* là các thao tác không đệ quy .

Ví dụ :

Cho dãy $\{ X_n \}$ xác định theo công thức truy hồi :

$X_0 = 1$; $X_n = n^2 X_O + (n-1)^2 X_1 + \; . \; . \; . \; + 2^2 X_{n-2} + 1^2 X_{n-1}$

+ Dạng hàm đệ quy tính X_n trên ngôn ngữ mã giả là :

$X_n \equiv$ if (n= 0) then return 1 ;

```
else  { tg = 0  ;
        for  i = 0   to   n-1  do    tg = tg + (n-i)^2 X_i   ;
        return   tg  ;
      }
```

\+ Dạng hàm đệ quy tính X_n trên ngôn ngữ Pascal là :

```
function   X( n :integer) : integer  ;
   var   i , tg : integer  ;
   begin
       if ( n= 0 )  then   X  :=   1
                 else
                   begin    tg = 0  ;
                    for  i: = 0  to  n-1  do  tg : = tg + sqr(n-i) *X(i)   ;
                       X  :=    tg  ;
                    end ;
     end ;
```

\+ Dạng hàm đệ quy tính X_n trên ngôn ngữ C++ là :

```
int    X(  int  n )   ;
   {  if ( n == 0 )  return  1 ;
         else  {    int  tg = 0  ;
                   for (int  i = 0 ; i<n ; i++ )   tg  = tg + sqr(n-i) *X(i);
                    return ( tg ) ;
                }
```

CHƯƠNG II
BÀI TOÁN ĐỆ QUY

I. CÁC NỘI DUNG CẦN LÀM ĐỂ TÌM GIẢI THUẬT ĐỆ QUY CHO MỘT BÀI TOÁN.

Để xây dựng giải thuật giải một bài toán có tính đệ quy bằng phương pháp đệ quy ta cần thực hiện tuần tự 3 nội dung sau :

- Thông số hóa bài toán .
- Tìm các trường hợp neo cùng giải thuật giải tương ứng .
- Tìm giải thuật giải trong trường hợp tổng quát bằng phân rã bài toán theo kiểu đệ quy.

1. Thông số hoá bài toán.

Tổng quát hóa bài toán cụ thể cần giải thành bài toán tổng quát (một họ các bài toán chứa bài toán cần giải),tìm ra các thông số cho bài toán tổng quát đặc biệt là nhóm các thông số biểu thị kích thước của bài toán – các thông số điều khiển (các thông số mà độ lớn của chúng đặc trưng cho độ phức tạp của bài toán , và giảm đi qua mỗi lần gọi đệ qui) .

Ví dụ : n trong hàm FAC(n) ; a , b trong hàm USCLN(a,b) .

2. Phát hiện các trường hợp suy biến (neo) và tìm giải thuật cho các trường hợp này.

Đây là các trường hợp suy biến của bài toán tổng quát , là các trương hợp tương ứng với các gía trị biên của các biến điều khiển (trường hợp kích thước bài toán nhỏ nhất), mà giải thuật giải không đệ qui (thường rất đơn giản).

Ví dụ :

FAC(1) =1 , USCLN(a,0) = a , SM(a[x:x] ≡∅ ,TSUM(a[m:m]) = a[m]

3. Phân rã bài toán tổng quát theo phương thức đệ quy.

Tìm phương án (giải thuật) giải bài toán trong trường hợp tổng quát bằng cách phân chia nó thành các thành phần mà hoặc có giải thuật không đệ quy hoặc là bài toán trên nhưng có kích thước nhỏ hơn.

Ví dụ : FAC(n) = n * FAC(n -1) .

Tmax(a[1:n]) = max(Tmax(a[1:(n-1)]) , a[n])

II. MỘT SỐ BÀI TOÁN GIẢI BẰNG GIẢI THUẬT ĐỆ QUY ĐIỂN HÌNH.

1. Bài toán tháp Hà Nội .

Truyền thuyết kể rằng : Một nhà toán học Pháp sang thăm Đông Dương đến một ngôi chùa cổ ở Hà Nội thấy các vị sư đang chuyển một chồng đĩa qúy gồm 64 đĩa với kích thước khác nhau từ cột A sang cột C theo cách :

- Mỗi lần chỉ chuyển 1 đĩa .
- Khi chuyển có thể dùng cột trung gian B .
- Trong suốt qúa trình chuyển các chồng đĩa ở các cột luôn được xếp đúng (đĩa có kích thước bé được đặt trên đĩa có kích thước lớn) .

Khi được hỏi các vị sư cho biết khi chuyển xong chồng đĩa thì đến ngày tận thế !.

Như sẽ chỉ ra sau này với chồng gồm n đĩa cần 2^n - 1 lần chuyển cơ bản (chuyển 1 đĩa).

Giả sử thời gian để chuyển 1 đỉa là t giây thì thời gian để chuyển xong chồng 64 đĩa sẽ là :

$$T = (2^{64} - 1) * t \ S = 1.84 * 10^{19} * t \ S$$

Với t = 1/100 s thì $T = 5.8*10^9$ năm = 5.8 tỷ năm .

Ta có thể tìm thấy giải thuật (dãy các thao tác cơ bản) cho bài toán một cách dễ dàng ứng với trường hợp chồng đĩa gồm 0, 1, 2, 3 đĩa . Với chồng 4 đĩa giải thuật bài toán đã trở nên phức tạp . Tuy nhiên giải thuật của bài toán lại được tìm thấy rất dễ dàng nhanh chóng khi ta khái quát số đĩa là n bất kỳ và nhìn bài toán bằng quan niệm đệ quy .

a) Thông số hóa bài toán .

Xét bài toán ở mức tổng quát nhất : chuyển n (n>=0) đĩa từ cột X sang cột Z lấy cột Y làm trung gian .

Ta gọi giải thuật giải bài toán ở mức tổng quát là thủ tục THN(n ,X ,Y,Z) chứa 4 thông số n,X,Y,Z ; n thuộc tập số tự nhiên N (kiểu nguyên không dấu); X ,Y,Z thuộc tập các ký tự (kiểu ký tự).

Bài toán cổ ở trên sẻ được thực hiện bằng lời gọi THN(64,A,B,C) .

Dễ thấy rằng : trong 4 thông số của bài toán thì thông số n là thông số quyết định độ phức tạp của bài toán (n càng lớn thì số thao tác chuyển đỉa càng nhiều và thứ tự thực hiện chúng càng khó hình dung) , n là thông số điều khiển .

b) Trường hợp suy biến và cách giải .

Với n =1 bài toán tổng quát suy biến thành bài toán đơn giản THN (1,X,Y,Z) : tìm dãy thao tác để chuyển chồng 1 đĩa từ cột X sang cột Z lấy cột Y làm trung gian . Giải thuật giải bài toán THN (1,X,Y,Z) là thực hiện chỉ 1 thao tác cơ bản : Chuyển 1 đĩa từ X sang Z (ký hiệu là Move (X , Z)) .

THN(1,X,Y,Z) ≡ { Move(X, Z) }

Chú ý : Hoàn toàn tương tự ta cũng có thể quan niện trường hợp suy biến là trường hợp n= 0 tương ứng với bài toán THN(0,X,Y,Z) : chuyển 0 đĩa từ X sang Z lấy Y làm trung gian mà giải thuật tương ứng là không làm gì cả (thực hiện thao tác rỗng) .

THN(0,X,Y,Z) ≡ { ϕ }

c) Phân rã bài toán :

Ta có thể phần rã bài toán TH N (k,X,Y,Z) : chuyển k đĩa từ cột X sang cột Z lấy cột Y làm trung gian thành dãy tuần tự 3 công việc sau :

+ Chuyển (k -1) đĩa từ cột X sang cột Y lấy cột Z làm trung gian :

THN (k -1,X,Z,Y) (bài toán THN với n = k-1,X= X , Y = Z , Z = Y)

+ Chuyển 1 đĩa từ cột X sang cột Z : Move (X, Z) (thao tác cơ bản).

+ Chuyển (k - 1) đĩa từ cột Y sang cột Z lấy cột X làm trung gian :

THN(k -1,Y,X,Z) (bài toán THN với n = k-1 , X = Y , Y = X , Z = Z) .

Vậy giải thuật trong trường hợp tổng quát (n > 1) là :

```
THN(n,X,Y,Z)   ≡   { THN (n -1,X,Z,Y) ;
                      Move ( X, Z ) ;
                      THN (n -1,Y,X,Z) ;
                      }
```

Với n đĩa thì cần bao nhiêu bước chuyển 1 đĩa? Thực chất trong thủ tục THN các lệnh gọi đệ qui chỉ nhằm sắp sếp trình tự các thao tác chuyển 1 đĩa

Số lần chuyển 1 đĩa được thực hiện là đặc trưng cho độ phức tạp của giải thuật .

Với n đĩa , gọi f(n) là số các thao tác chuyển _một_đĩa .

Ta có : $f(0) = 0$.

$f(1) = 1$.

$f(n) = 2f(n-1) + 1$ với $n > 0$

Do đó : $f(n) = 1 + 2 + 2^2 + \quad + 2^{n-1} = 2^n - 1$

Để chuyển 64 đĩa cần $2^{64} - 1$ bước hay xấp xỉ 10^{20} bước . Cần khoảng 10 triệu năm với một máy tính nhanh nhất hiện nay để làm việc này .

d) Chương trình con mã hóa giải thuật THN trong NNLT Pascal :

```
procedure   THN (n : integer ; X,Y,Z : char)
          begin
               if   n > 0   then   begin
                                    THN (n-1 ,X,Z,Y) ;
                                     Move( X, Z);
                                     THN (n-1 ,Y,X,Z);
                                  end ;
            end ;
```

(Lấy trường hợp chuyển n = 0 làm trường hợp neo)

```
Hoặc :          procedure   THN (n : integer ; X,Y,Z : char)
                    begin
                        if (n = 1)  then    Move(X, Z)
                                 else   begin
                                           THN (n-1 ,X,Z,Y ) ;
                                           Move(X, Z );
                                           THN (n -1 ,Y,X,Z );
                                        end ;
                        end;
```

(Lấy trường hợp chuyển n = 1 làm trường hợp neo)

Với thủ tục Move(X, Y) mô tả thao tác chuyển 1 đĩa từ cột X sang cột Y được viết tuỳ theo cách thể hiện thao tác chuyển .

e) Chương trình con mã hóa giải thuật THN trong NNLT C++ :

Trong C++ hàm con thực hiện giải thuật THN có dạng :

```
void  THN( int  n , char  X,Y,Z)
    { if(n > 0)
           { THN(n -1,X,Z,Y ) ;
             Move ( X , Z ) ;
             THN(n - 1,Y,X,Z ) ;
             }
      return ;
      }
```

hoặc :

```
void  THN( int n , char  X,Y,Z)
      { if(n = = 1)   Move ( X , Z ) ;
         else
           { THN(n -1,X,Z,Y ) ;
              Move ( X, Z ) ;
              THN(n - 1,Y,X,Z ) ;
             }
        return ;
       }
```

2. Bài toán chia thưởng.

Có 100 phần thưởng đem chia cho 12 học sinh giỏi đã được xếp hạng. Có bao nhiêu cách khác nhau để thực hiện cách chia?

Ta thấy ngay rằng việc tìm ra lời giải cho bài toàn sẻ không dễ dàng nếu ta không tìm ra cách thích hợp để tiếp cận với nó. Ta sẽ tìm giải thuật giải bài toàn bằng phương pháp đệ quy.

a) Thông số hóa.

Ta sẽ giải bài toán ở mức độ tổng quát : Tìm số cách chia m vật (phần thưởng) cho n đối tượng (học sinh) có thứ tự .

Gọi PART là số cách chia khi đó PART là hàm của 2 biến nguyên m , n (PART(m ,n)) .

Ta mã hoá n đối tượng theo thứ tự xếp hạng 1, 2 , 3 , . . . n ; Si là số phần thưởng mà học sinh i nhận được .

Khi đó các điều kiện ràng buộc lên cách chia là :

$S_i >= 0$

$S_1 >= S_2 >= \quad >= S_n$.

$S_1 + S_2 + \quad + S_n = m$

Ví dụ :

Với m = 5 , n = 3 ta có 5 cách chia sau :

5 0 0
4 1 0
3 2 0
3 1 1
2 2 1

Tức là PART(5,3) = 5

b) Các trường hợp suy biến :

+ m = 0 thì sẻ có duy nhất 1 cách chia : mọi học sinh đều nhận được 0 phần thưởng .

Vậy : PART(0 , n) = 1 với mọi n

+ n = 0 , m <> 0 thì sẽ không có cách nào để thực hiện việc chia .

Vậy : PART(m , 0) = 0 với mọi m <> 0 .

(ta có thể thay trường hợp neo PART(m ,0) = 0 hoặc trường hợp neo PART(m , 1) = 1)

c) Phân rã bài toán trong trường hợp tổng quát :

+ m < n khi số phần thương m nhỏ hơn số học sinh n thì n - m học sinh xếp cuối sẽ luôn không nhận được gì cả trong mọi cách chia .

Vậy :

khi n > m thì PART(m , n) = PART(m , m) .

+ Trong trường hợp m >= n : số vật chia (phần thưởng) lớn hơn hoặc bằng số học sinh (đối tượng) ta phân các cách chia làm 2 nhóm :

* Nhóm thứ nhất không dành cho học sinh xếp cuối cùng phần thưởng nào cả

($S_n = 0$) . Số cách chia này sẽ bằng số cách chia m phần thương cho n -1 học sinh .

Tức là : Số cách chia trong nhóm thứ nhất = PART(m , n -1) .

* Nhóm thứ 2 có phần cho người cuối cùng ($S_n > 0$) . Dễ thấy rằng số cách chia của nhóm này bằng số cách chia m - n phần thương cho n học sinh (vì phương thức chia mà tất cả học sinh đều nhận được phần thưởng có thể thực hiện bằng cách : cho mỗi người nhận trước 1 phần thưởng rồi mới chia).

Tức là : Số cách chia trong nhóm thứ 2 = PART(m - n , n) .

Vậy : với m>= n PART(m , n) = PART(m , n -1) + PART(m - n , n)

d) Dạng mã giả của hàm PART(m , n)

```
PART(m , n ) = if(m = 0 ) then return 1 ;
               else if( n = 1 ) then return 1 ;
                    else if(m < n ) then return PART(m , m) ;
                         else return ( PART(m , n -1) + PART(m - n , n ))
```

e) Dạng hàm PART trong NNLT Pascal

```
Function PART(m , n : integer ) : integer ;
    Begin
      if ( (m = 0) or ( n = 1) ) then PART := 1
             else if(m < n) then PART := PART(m , m )
                   else PART := PART(m , n -1 ) + PART(m - n , n) ;
     End ;
```

g) Dạng hàm PART trong NN LT C++

```
int PART( int m , int n )
  { if ((m == 0 ) || (n == 0) ) return 1 ;
          else if(m < n ) retrun ( PART(m , m )) ;
               else return ( PART(m , n -1 ) + PART( m -n , n ) ) ;
    }
```

3. Bài toán tìm tất cả các hoán vị của một dãy phần tử.

Bài toán : Xuất tất cả các hoán vị của dãy A .

Ví dụ : Với dãy A gồm N = 3 phần tử A[1] = a , A[2] = b , A[3] = c thì bài toán bắt phải xuất 6 hoán vị có thể của A :

a b c   a c b   c b a

b a c   c a b   b c a

Với dãy A gồm N = 4 phần tử A[1] = 1 , A[2] = 2 , A[3] = 3 , A[4] =4 thì bài toán bắt phải xuất 24 hoán vị có thể của A :

1 2 3 4   1 2 4 3   1 4 3 2   4 2 3 1

2	1	3	4	2	1	4	3	4	1	3	2	2	4	3	1
1	3	2	4	1	4	2	3	1	3	4	2	4	3	2	1
3	1	2	4	4	1	2	3	3	1	4	2	3	4	2	1
3	2	1	4	4	2	1	3	3	4	1	2	3	2	4	1
2	3	1	4	2	4	1	3	4	3	1	2	2	3	4	1

a) Thông số hóa bài toán .

Gọi HV(v, m) (với v : array[1 . . N] of T , m :integer ; m ≤ N ; T là một kiểu dữ liệu đã biết trước) là thủ tục xuất tất cả các dạng khác nhau của v có được bằng cách hoán vị m thành phần đầu của dãy v

Ví dụ : N = 4 , A[1] = 1 , A[2] = 2 , A[3] = 3 , A[4] = 4 thì lời gọi HV(A ,3) xuất tất cả hoán vị của A có được bằng cách hoán vị 3 phần tử đầu (có 6 h vị) :

1	2	3	4	1	3	2	4	3	2	1	4
2	1	3	4	3	1	2	4	2	3	1	4

Để giải bài toán đặt ra ban đầu ta gọi HV(A,N)).

b) Trường hợp neo.

Vơi m = 1 : HV(v,1) là thủ tục giải bài toán xuất tất cả các dạng của v có được bằng cách hoán vị 1 phần tủ đầu . Vậy HV(v,1) là thủ tục xuất v.

HV(v,1) ≡ print(v) ≡ for k:= 1 to N do write(v[k])

c) Phân rã bài toán.

Ta có thể tìm hết tất cả các hoán vị m phần tử đầu của vector V theo cách sau :

- Giữ nguyên các phần tử cuối V[m] , . . . ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .
- Đổi chổ V[m] cho V[m-1] ,giữ nguyên các phần tử cuối V[m],... ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .
- Đổi chổ V[m] cho V[m-2] ,giữ nguyên các phần tử cuối V[m],…. ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .

. . . .

. . . .

- Đổi chổ V[m] cho V[2] ,giữ nguyên các phần tử cuối V[m], . .. ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .
- Đổi chổ V[m] cho V[1] ,giữ nguyên các phần tử cuối V[m], . . . ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .

Vậy :

```
HV(V,m) ≡ {  SWAP( V[m],V[m] )      ; HV(V,m – 1) ;
             SWAP( V[m],v[m-1] ) ; HV(V,m – 1) ;
             SWAP( V[m],v[m-2 ] ) ; HV(V,m – 1) ;
```

```
                    . . . . . . . . . . . . . . . . . . . . . . . . . . . .
                      . . . . . . . . . . . . . . . . . . . . . . . . . . . .
                    SWAP (V[m],v[2] )      ;  HV(V,m – 1) ;
                     SWAP( V[m],v[1] )      ; HV(V,m – 1) ;
                    }
```

(SWAP(x , y) là thủ tục hoán đổi giá trị của 2 đối tượng dữ liệu x ,y)

Vậy :

```
HV(V , m )   ≡   for  k :=  m   downto  1  do  begin
                                                SWAP( V[m], V[k] ) ;
                                                 HV(V,m – 1) ;
                                              end ;
```

d) Thủ tục hoán vị trên NNLT Pascal.

```
      . . . . . . . . . . . . . . . . . .
const   size = Val ; (* Val là hằng gía trị *)
type  vector = array[1.  . size] of  typebase; (* typebase là một kiểu dữ liệu có thứ tự *)
   . . . . . . . . . . . . . . . . . . . . . .
procedure  Swap( var  x , y : typebase ) ;
     var     t : typebase ;
      begin
            t := x ; x := y ; y := t ;
       end ;
     . . . . . . . . . . . . . . . . . . . . . . . . . .
 procedure  print( A : vector ) ;
     var     i : integer ;
      begin
             for   i:= 1   to   size  do   write( A[i] : 3 );
              writeln ;
       end ;
     . . . . . . . . . . . . . . . . . . . . . . . . . .

Procedure    HV( V : vec tor ; m :integer ) ;
    var    k : integer ;
      begin
         if (m = 1 ) then  print(V)
                  else
                     for  k := m  downto  1 do  begin
                                                  Swap(V[m] , V[k]);
                                                  HV(V , m – 1) ;
                                                end ;
      end ;
```

e) Thủ tục hoán vị trên NNLT C++ .

```
        . . . . . . . . . . . . . . . . . .
const   size = Val ;  //  Val là hằng gía trị
typedef        typebase vector[size] ; // typebase là một kiểu dữ liệu có thứ tự
     . . . . . . . . . . . . . . . . . . . . . .
void  Swap(  typebase & x , typebase& y)
   { typebase      t  ;
        t = x ; x = y ; y = t ;
     }
    . . . . . . . . . . . . . . . . . . . . . . . . .  .
void  print( const vector &A)
      { for(int  j= 0 ; j <size ; j++ ) cout<< A[j] ;
          cout << endl ;
          }
    . . . . . . . . . . . . . . . . . . . . . . . . .  .

void  HV( const vector &V , int  m)
      {  if (m == 1 )   print( V );
           else   for(int  k = m-1 ; k > = 0 ; k-- )
           {  swap(V[m-1] ,V[k] ) ;
              HV(V,m-1) ;
              }
        }
```

4. Bài toán sắp xếp mảng bằng phương pháp trộn (Sort-Merge).

Ý tưởng : Để sắp xếp 1 danh sách gồm n phần tử bằng phương pháp trộn người ta chia danh sách thành 2 phần (tổng quát là nhiều phần) , sắp xếp từng phần, rồi trộn chúng .

Bài toán : sắp theo thứ tự không giảm mảng a : VectorT bằng phương pháp trộn.
(VectorT = array[1 . . size] of T).

a) Thông số hoá:

Bài toán được khái quát thành sắp xếp một dãy con của dãy V : VectorT từ chỉ số m đến chỉ số n với 1 <= m <= n <= size . Ta đặt tên cho bài toán ở dạng tổng quát là : SM(V,m,n).

Bài toán ban đầu : sắp dãy A sẻ được thực hiện bằng lời gọi : SM(A ,1,size).

b) Trường hợp tầm thường:

Đó là khi n = m (dãy sắp chỉ có 1 phần tử), khi đó không cần làm gì cả (thao tác rỗng) .

c) Phân rã trường hợp tổng quát :

Khi n > m ta thực hiện các công việc sau :

+ Chia dãy : a[m] ,a[m+1], . . . , a[n] thành 2 dãy con
a[m] , . . , a[l] và a[l+1] , . . . , a[n]

+ Sắp xếp từng dãy con thành các dãy có thứ tự theo giải thuật SM .

+ Trộn 2 dãy con có thứ tự lại thành dãy a[m] ,. . . , a[n] mới có thứ tự .

Để thực hiện việc trộn hai dãy có thứ tự thành một dãy có thứ tự ta sẽ dùng một thủ tục không đệ quy Merge(m , l , n) . Ta cần chọn l để được 2 dãy con giảm hẵn kích thước so với dãy ban đầu , tức là chọn l : m < l < l+1 < n .

Thương chọn l là phần tử "giữa " : l = (m + n) div 2 .

Thủ tục Sort_Merge(m,n) trên mảng V : VectorT viết trên ngôn ngữ PASCAL có dạng :

```
procedure  SM (var d: VectorT ; m,n: index);
 var  l : index ;
 begin
     if  n>m  then
            begin
              l := (m+n) div 2;
              SM (d,m,l) ;
              SM (d,l+1,n) ;
              Merge (d,m,l,n)  ;
           end  ;
  end ;
```

Trong đó SM là thủ tục trộn 2 dãy tăng để được một dãy tăng.

Để sắp mảng A (dãy A[1:size]) ta gọi SM(A ,1,size)

5. Bài toán tìm nghiệm xấp xỉ của phương trình f(x)=0 .

Bài toán : Hàm f(x) liên tục trên đoạn $[a_o,b_o]$, tìm một nghiệm xấp xỉ với độ chính xác ε trên $[a_o,b_o]$ của phương trình f(x) = 0.

Ý tưởng của giải thuật :

- Trường hợp neo : $b_o - a_o < \varepsilon$

+ Nếu $f(a_o).f(b_o) \le 0$ thì hàm f có nghiệm trên $[a_o,b_o]$ .Và vì ta đang tìm nghiệm xấp xỉ với độ chính xác ε nên a_o là nghiệm xấp xỉ cần tìm .

+ Nếu $f(a_o).f(b_o) > 0$ thì ta xem như không có nghiệm xấp xỉ trên đoạn xét.

- Trương hợp $b_o - a_o \ge \varepsilon$ thì chia đôi đoạn $[a_o,b_o]$ rồi tìm lần lượt nghiệm trên từng đoạn con : đoạn con trái, đoạn con phải .

Ta sẽ xây dựng một hàm đệ qui trả về giá trị là nghiệm xấp xỉ của f (nếu có),hay một hằng E (đủ lớn) nếu f không có nghiệm xấp xỉ trên $[a_o,b_o]$.

a) Thông số hoá:

Xét hàm ROOT với 3 thông số là g , a,b ,(ROOT(g,a,b)) trả về giá trị nghiệm xấp xỉ ε của phương trình g(x) =0 trên đoạn [a,b] hoặc giá trị C nếu phương trình xét không có nghiệm xấp xỉ . Để giải bài toán ban đầu ta gọi hàm ROOT(f,a_o,b_o) .

b) Trường hợp tầm thường:

đó là khi b - a < epsilon .
Khi đó :

```
if ( g(a).g(b) ) <=  0  then  ROOT(g,a,b) = a ; // a là nghiệm xấp xỉ
                            else   ROOT(g,a,b) = E ; // không có nghiệm  xấp xỉ
```

c) Phân rã trường hợp tổng quát:

khi b - a >= ε ta phân [a,b] làm 2 đoạn [a,c] và [c,b] với c = (a + b) / 2.

- Nếu ROOT(g , a ,c) < E thì ROOT(g , a , b) = ROOT(g ,a ,c) (bài toán tìm nghiệm trên đoạn [a,c]) .
- còn không thì ROOT(g , a , b) = ROOT(g ,c ,b) (bài toán tìm nghiệm trên đoạn [c ,b]) .

d) Hàm tìm nghiệm xấp xỉ trên NN Pascal có dạng:

```
  const    epsilon =     ;
                 E  =    ;
 Function ROOT(a,b :real ) :real ;
 var     c , R : real ;
 begin
    if ((b-a) < epsilon ) then  if ( f(a)*f(b) <= 0 ) then  ROOT  :=  a
                                                    else  ROOT  :=  L
                          else
                           begin
                             c := (a + b)/2 ;
                             if  ( ROOT(a ,c ) < E )   then    ROOT := ROOT(a,c)
                                                       else    ROOT := ROOT(c,b)
                                end;
```

e) Chương trình con tìm nghiệm xấp xỉ trong NN LT C++

```
const double  epsilon  =       ;
const  double        E   =       ;
 double  ROOT(double a , double b )
   { if((b - a) < epsilon )   if(f(a)*f(b) <=  epsilon   return (a ) ;
                              else    return ( L ) ;
       else
         { double  c = (a + b ) / 2  ;
```

```
double   R = ROOT(a,c) ;
if( R< E )   return R  ;
                else  return ( ROOT(c , b) ) ;
   }
}
```

CHƯƠNG III
KHỬ ĐỆ QUY

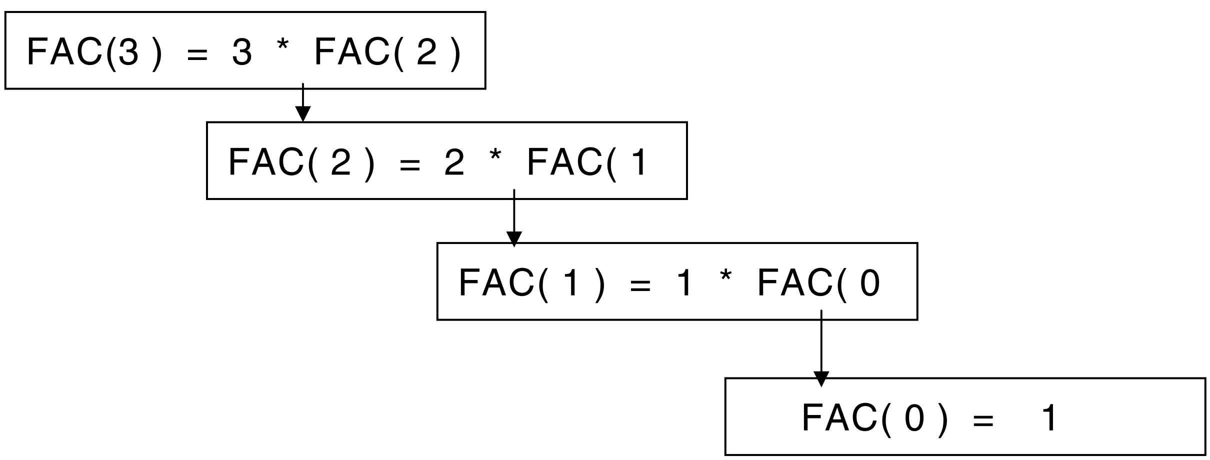
I. CƠ CHẾ THỰC HIỆN GIẢI THUẬT ĐỆ QUY.

Trạng thái của tiến trình xử lý một giải thuật ở một thời điểm được đặc trưng bởi nội dung các biến và lệnh cần thực hiện kế tiếp. Với tiến trình xử lý một giải thuật đệ qui ở từng thời điểm thực hiện, con cần lưu trữ cả các trạng thái xử lý đang còn dang dở .

a) Xét giải thuật đệ quy tính giai thừa:

FAC (n) ≡ if(n = 0) then retrun 1 ;
else retrun (n * FAC (n – 1)) ;

Sơ đồ quá trình tính gía trị 3 ! theo giải thuật đệ quy :



Khi thực hiện lời gọi FAC (3) sẻ phát sinh lòi gọi FAC (2) , đồng thời phải lưu giữ thông tin trạng thái xử lý còn dang dỏ (FAC (3) = 3 * FAC (2)) . Đến lượt mình lời gọi FAC (2) lại làm phát sinh lời gọi FAC (1) ,đồng thời vẩn phải lưu trử thông tin trạng thái xử lý còn dang dở (FAC (2) = 2 * FAC (1)) ,.. . Cứ như vậy cho tới khi gặp lời gọi
trường hợp neo (FAC (0) = 1) .

Tiếp sau qúa trình gọi là một qúa trình xử lý ngược được thực hiện :

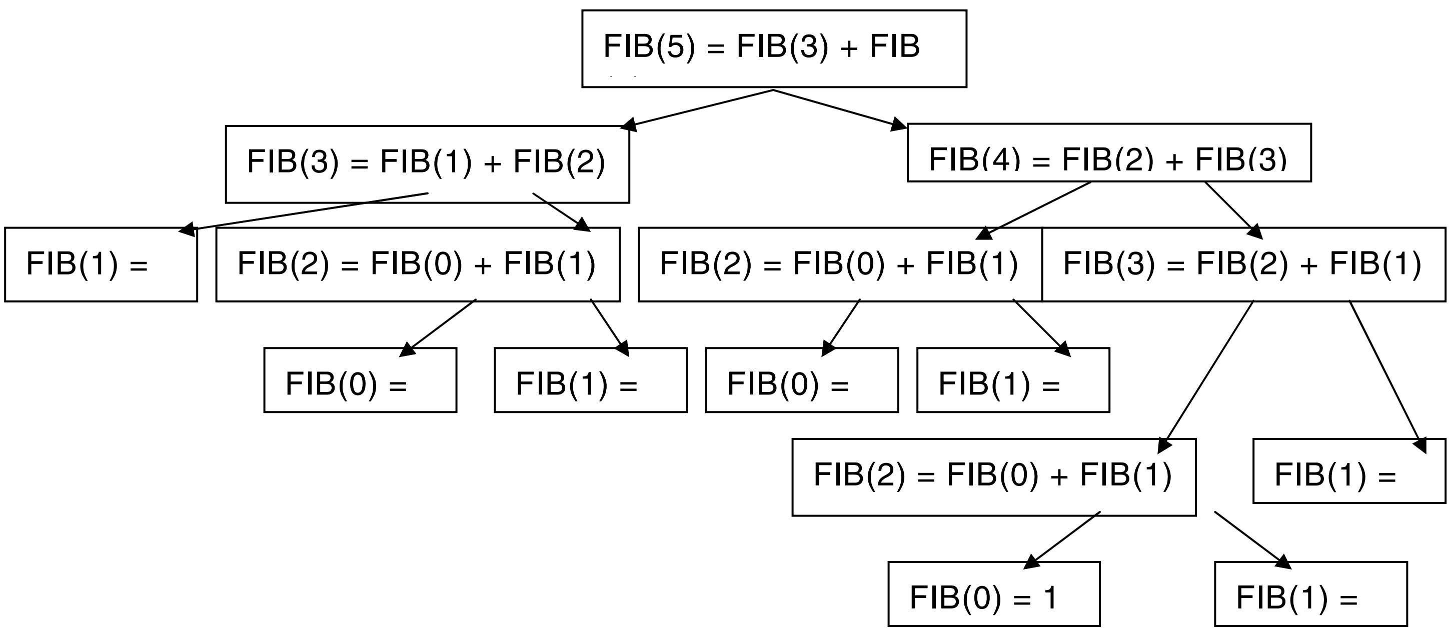
- Dùng giá trị FAC (0) để tính FAC (1) theo sơ đồ xử lý còn lưu trử .
- Dùng giá trị FAC (1) để tính FAC (2) theo sơ đồ xử lý còn lưu trử .
- Dùng giá trị FAC (2) để tính FAC (3) theo sơ đồ xử lý còn lưu trử .

Đồng thời với qúa trình xử lý ngược là qúa trình xóa bỏ các thông tin về giải thuật xử lý trung gian (qúa trình thu hồi vùng nhớ) .

b) Xét giải thuật đệ quy tính giá trị hàm FIBONACCI .

FIB(n) ≡ if ((n = 0) or (n = 1)) then return 1 ;
else return (FIB(n - 1) + FIB(n - 2)) ;

Sơ đồ tính FIB(5) :



c) Xét thủ tục đệ quy tháp Hà Nội THN (n , X , Y , Z)

```
THN (n : integer ; X ,Y , Z : char)
      ≡     if  (n > 0 )  then
                       { THN(n-1,X ,Z ,Y) ;
                         Move(X, Z) ;
                         THN(n-1,Y,X,Z) ;
                       }
```

Để chuyển 3 đĩa từ cột A sang cột C dùng cột B làm trung gian ta gọi : THN (3,A,B,C)
Sơ đồ thực hiện lời gọi THN (3,A,B,C) là :

Lời gọi c/0	**Lới gọi c/1**	**Lời gọi c/2**	**Lời gọi c/3**
			THN(0,A,C,B)
		THN(1,A,B,C)	A ---> C
			THN(0,B,A,C)
	THN(2,A,C,B)	A ---> B	
			THN(0,C,B,A)
		THN(1,C,A,B)	C --->B
			THN(0,A,C,B)
THN(3,A,B,C)	A ---> C		
			THN(0,B,A,C)
		THN(1,B,C,A)	B ---> A
			THN(0,C,B,A)
	THN(2,B,A,C)	B ---> C	
			THN(0,A,C,B)
		THN(1,A,B,C)	A ---> C
			THN(0,B,A,C)

Với THN(0 ,X , Y , Z) là trường hợp neo tương ứng với thao tác rỗng .

X ------> Y là thao tác chuyển 1 đĩa từ cột X sang cột Y (MOVE(X,Y)).

Các bước chuyển đĩa sẻ là :

A --> C ; A --> B ; C --> B ; A --> C ; B --> A ; B --> C ; A --> C ;

Lời gọi cấp 0 :

THN(3 , A , B , C) sẻ làm nảy sinh hai lời gọi cấp 1 : THN (2 ,A, C, B) ; THN (2 , B , A , C) cùng với các thông tin của qúa trình xử lý còn dang dở .

Các lời gọi cấp 1 :

THN(2 , A , C , B) , THN (2 , B , A ,C) sẻ làm nảy sinh các lời gọi cấp 2 : THN (1 ,A, B, C) ; THN (1, C , A , B) ; THN (1 ,B, C, A) ; THN (1, A , B , C) ; cùng với các thông tin của qúa trình xử lý còn dang dở .

Các lời gọi cấp 2 :

THN(1 ,A, B, C) ; THN(1, C , A , B) ; THN(1 ,B, C, A) ; THN(1, A , B , C) ; sẻ làm nảy sinh các lời gọi cấp 3 dạng : THN(0 ,X, Y, Z) (thao tác rỗng tương ứng với trường hợp suy biến); cùng với các thông tin của qúa trình xử lý còn dang dở .

Quá trình gọi dừng lại khi gặp trường hợp suy biến .

Qúa trình xử lý ngược với quá trình gọi bắt đầu khi thực hiện xong các trường hợp neo nhằm hoàn thiện các bước xử lý con dang dở song song với quá trình hoàn thiện các lời gọi là qúa trình loại bỏ các lưu trữ thông tin giải thuật trung gian.

Do đặc điểm của qúa trình xử lý một giải thuật đệ quy là : việc thực thi lời gọi đệ quy sinh ra lời gọi đệ quy mới cho đến khi gặp trường hợp suy biến (neo) cho nên để thực thi giải thuật đệ quy cần có cơ chế lưu trữ thông tin thỏa các yêu cầu sau :

+ Ở mỗi lần gọi phải lưu trữ thông tin trạng thái con dang dở của tiến trình xử lý ở thời điểm gọi. Số trạng thái này bằng số lần gọi chưa được hoàn tất .

+ Khi thực hiện xong (hoàn tất) một lần gọi, cần khôi phục lại toàn bộ thông tin trạng thái trước khi gọi .

+ Lệnh gọi cuối cùng (ứng với trương hợp neo) sẽ được hoàn tất đầu tiên , thứ tự dãy các lệnh gọi được hoàn tất ngược với thứ tự gọi, tương ứng dãy thông tin trạng thái được hồi phục theo thứ tự ngược với thứ tự lưu trữ .

Cấu trúc dữ liệu cho phép lưu trữ dãy thông tin thỏa 3 yêu cầu trên là cấu trúc lưu trữ thỏa luật LIFO (Last In Firt Out) . Một kiểu cấu trúc lưu trữ thường được sử dụng trong trường hợp này là cấu trúc chồng (stack).

Với một chồng S thường cho phép chúng ta thực hiện các thao tác sau trên nó :

- Thủ tục Creatstack(S) : Tạo chồng S rỗng .
- Thủ tục Push(x,S) : Lưu trữ thêm dữ liệu x vào đĩnh stack S
  (x là dữ liệu kiểu đơn giản giản hoặc có cấu trúc)
- Thủ tục Pop(x,S) : Lấy giá trị đang lưu ở đĩnh S chứa vào trong đối tượng dữ liệu x và loại bỏ giá trị này khỏi S (lùi đỉnh S xuống một mức) .
- Hàm Empty(S) : (kiểu boolean) Kiểm tra tính rỗng của S : cho giá trị đúng nếu S rỗng , sai nếu S không rỗng .

Cài đặt cụ thể của S có thể thực hiện bằng nhiều phương pháp phụ thuộc vào từng ngôn ngữ lập trình và từng mục đích sử dụng cụ thể .

Ví dụ :

Cài đặt (bằng cấu trúc mảng) chồng S mà mỗi phần tử là một đối tượng dữ liệu thuộc kiểu T trong PASCAL như sau :

```
Const sizestack =   . . . ;
 Type   stackType  =  record
                          St : array [1 . . sizestack ]  of   T  ;
                         Top : 0 . .  sizestack  ;
                     end  ;
```

Thủ tục Creatstack(S) : tạo chồng S rỗng :

```
Procedure   Creatstack( var  S : StackType )
     Begin
          S.Top := 0 ;
     End;
```

Thủ tục Push(x,S) : Chèn - Lưu trữ thêm dữ liệu x vào đĩnh stack S
(x là dữ liệu kiểu đơn giản giản hoặc có cấu trúc)

```
Procedure   Push( var  S : StackType ; x : T) ;
     Begin
```

```
            S.St[S.Top +1] :=  x ;
            S.Top   :=    S.Top + 1 ;
        End;
```

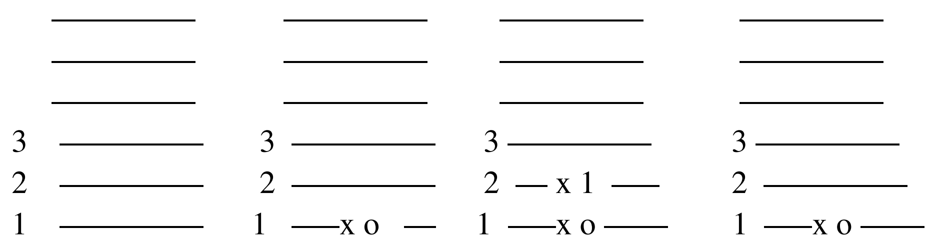
Thủ tục Pop(x,S) : Xóa - Lấy giá trị đang lưu ở đỉnh S chứa vào trong đối tượng dữ liệu x và loại bỏ giá trị này khỏi S (lùi đỉnh S xuống một mức) .

```
Procedure   Pop( var  S : StackType ; var  x : T ) ;
        Begin
            x  :=  S.St[S.Top] ;
            S.Top   :=    S.Top  -  1 ;
        End;
```

Hàm Empty(S) : (Hàm boolean) Kiểm tra tính rỗng của Stack S

```
Function   Empty( S : StackType ) : boolean ;
    Begin
        Empty  :=  ( S.Top = 0 ) ;
    End ;
```

Mô hình stack S và tác dụng các thao tác trên nó .



Createstack(S) ;	Push(S, x_o) ;	Push(S,x_1) ;	pop(S,y)
(S.top = 0)	S.St[1] := x_o	S.St[2] := x_1	y := x_1
S.top := 1	S.top := 2	S.Top := 1 ;	

NNLT PASCAL và C++ thực hiện được cơ chế đệ qui nhờ trong quá trình biên dịch, phần mềm ngôn ngữ tự động phát sinh ra cấu trúc stack để quản lý các lệnh gọi chương trình con. Khi một lệnh gọi chương trình con thực hiện, các biến địa phương (gồm cả các thông số) sẽ được cấp phát vùng nhớ mới ở đỉnh stack. Nhờ vậy các tác động địa phương của thủ tục sẽ không làm thay đổi các trạng thái xử lý còn dang dở.

II. TỔNG QUAN VỀ VẤN ĐỀ KHỬ ĐỆ QUY.

Đệ quy là phương pháp giúp chúng ta tìm giải thuật cho các bài toán khó . Giải thuật giải bài toán bằng đệ quy thường rất đẹp (gọn gàng, dễ hiểu ,dễ chuyển thành

chương trình trên các NNLT) . Nhưng như đã chỉ ra ở trên việc xử lý giải thuật đệ quy lại thường gây khó khăn cho máy tính (tốn không gian nhớ và thời gian xử lý), hơn nữa không phải mọi NNLT đều cho phép mã hóa giải thuật đệ quy (ví dụ : FORTRAN) . Vì vậy việc thay thế một chương trình đệ quy (có chứa chương trình con đệ quy) bằng một chương trình không đệ quy cũng là một vấn đề được quan tâm nhiều trong lập trình .

Một cách tổng quát người ta đã chỉ ra rằng : Mọi giải thuật đệ quy đều có thể thay thế bằng một giải thuật không đệ quy . Vấn đề còn lại là kỹ thuật xây dựng giải thuật không đệ quy tương ứng thay thế giải thuật đệ quy . Rất đáng tiếc việc xậy dựng giải thuật không đệ quy thay thế cho một giải thuật đệ quy đã có lại là một việc không phải bao giờ cũng đơn giản và đến nay vẫn chưa có giải pháp thỏa đáng cho trường hợp tổng quát.

Sơ đồ để xây dựng chương trình cho một bài toán khó khi ta không tìm được giải thuật không đệ quy thường là :

+ Dùng quan niệm đệ quy để tìm giải thuật cho bài toán .
+ Mã hóa giải thuật đệ quy .
+ Khử đệ quy để có được một chương trình không đệ quy .

Tuy nhiên do việc khử đệ quy không phải bao giờ cũng dễ và vì vậy trong nhiều trường hợp ta cũng phải chấp nhận sư dụng chương trình đệ quy .

III. CÁC TRƯỜNG HỢP KHỬ ĐỆ QUY ĐƠN GIẢN.

1. Các trường hợp khử đệ quy bằng vòng lặp .

a) Hàm tính gía tri của dãy dữ liệu mô tả bằng hồi quy .

a_1) Ý tưởng dẫn dắt :

Xét một vòng lặp trong đó sử dụng 1 tập hợp biến W = (V , U) gồm tập hợp U các biến bị thay đổi trong vòng lặp và V là các biến còn lại.

Dạng tổng quát của vòng lặp là :

W := W_o ; { W_o = (U_o,V_o) }
while C(U) do U := g(W) (3.1.1)

Gọi U_o là trạng thái của U ngay trước vòng lặp , U_k với k >0 là trạng thái của U sau lần lặp thứ k (giả sử còn lặp đến lần k) .

Ta có :

U_o mang các giá trị được gán ban đầu
U_k = g(W) = g(U_{k-1} , V_o) = f(u_{k-1}) với k = 1 .. n (3.1.2)

Với n là lần lặp cuối cùng , tức C(U_k) đúng với mọi k < n , C(U_n) sai

Sau vòng lặp W mang nội dung (U_n ,V_o) .

Ta thấy : để tính gía trị dãy được định nghĩa bởi quan hệ hồi quy dạng (3.1.2) ta có thể dùng giải thuật lặp mô tả bởi đoạn lệnh (3.1.1) .

a_2) Giải thuật tính gía trị của dãy hồi quy thường gặp dạng :

f(n) = C khi $n = n_o$ (C là một hằng)
= g(n,f(n -1)) khi n > no

Ví dụ :

Hàm giai thừa FAC (n) = n ! = 1 khi n = 0
= n * FAC(n - 1) khi n > 0

Tổng n số đầu tiên của dãy đan dấu sau :

$S_n = 1 - 3 + 5 - 7 .. + (-1)^{n+1} * (2n-1)$

S(k) = 1 khi k =1
= $S(k-1) + (- 1)^{k+1} *(2*k-1)$ với k > 1

- Giải thuật đệ quy tính giá trị f(n)

```
f(n) = if(n = no) then return C ;
                  else return (g(n,f(n -1)) ;
```

- Giải thuật lặp tính giá tri f(n)

```
k := no ; F := C ;
  { F = f(no) }
 While( k < n ) do begin
                       k := k +1 ;
                      F := g(k,F ) ;
                    end ;                          }
        { F = f(n) }
```

Hoặc :

```
F := C ;
 For k := no to n -1 do begin
                          k := k + 1 ;
                          F := g(k,F) ;
                        end ;
```

Trong trường hợp này :

W = U = (k ,F)

$W_o = U_o = (n_o, C)$

C(U) = (k < n)

f(W) = f(U) = f(k,F) = (k+1,g(k,F)))

Ví dụ 1: Hàm tính FAC(n) = n! không đệ quy

+ Trong NN LT PASCAL

```
Function FAC ( n : integer ) : longint ;
  var k : integer ;
      F : longint ;
  Begin
     F := 1 ; k := 0 ;
     while (k < n ) do begin
```

```
                        k := k + 1 ;
                         F := F * k ;
                      end ;
          FAC := F ;
      end ;
```

hoặc :

```
  Function  FAC ( n : integer ) : longint ;
   var  k  : integer ;
        F  :  longint ;
   Begin
       F := 1 ;
       For  k:= 1 to  n do   F := F * k ;
       FAC := F ;
    end ;
```

+ Trong NN LT C++

```
 long int  FAC ( int  n )
   {   int  k = 0 ;
      long int  F = 1 ;
      while ( k < n )  F =  ++k * F ;
      return (F) ;
      }
```

Hoặc :

```
 long int  FAC ( int  n )
   {    long int  F = 1 ;
         for ( int k = 1; k <= n ; k++)  F  =  k * F ;
      return (F) ;
      }
```

Ví du 2 : Dạng hàm S_n không đệ quy

+ trên NN LT Pascal :

```
  Function  S(n : integer ) : integer ;
    var  k ,tg :  integer ;
    Begin
         k := 1 ; tg := 1 ;
         while ( k < n ) do  begin
                             k := k + 1 ;
                             if odd (k)   then  tg := tg + (2 * k - 1 )
                                          else  tg := tg - (2 * k - 1 ) ;
                           end ;
          S :=  tg ;
     end ;
```

+ Trong NN LT C++

```
int  S ( int  n )
   { int  k = 1 , tg = 1 ;
   while ( k < n )  { k ++ ;
                      if (k%2)  tg +  =  2 * k - 1 ;
                        else     tg  -  =   2 * k + 1 ;
                    }
    return ( tg ) ;
  }
```

b) Các thủ tục đệ qui dạng đệ qui đuôi.

Xét thủ tục P dạng :

```
P(X) ≡   if  B(X)   then   D(X)
                    else   { A(X)  ;
                             P(f(X)) ;
                           }
```

Trong đó : X là tập biến (một hoặc một bộ nhiều biến).

P(X) là thủ tục đệ quy phụ thuộc X

A(X) ; D(X) là các nhóm thao tác (lệnh) không đệ quy

f(X) là hàm biến đổi X

Xét qúa trình thi hành P(X) :

gọi P_o là lần gọi P thứ 0 (đầu tiên) P(X)

P_1 là lần gọi P thứ 1 (lần 2) P(f(X))

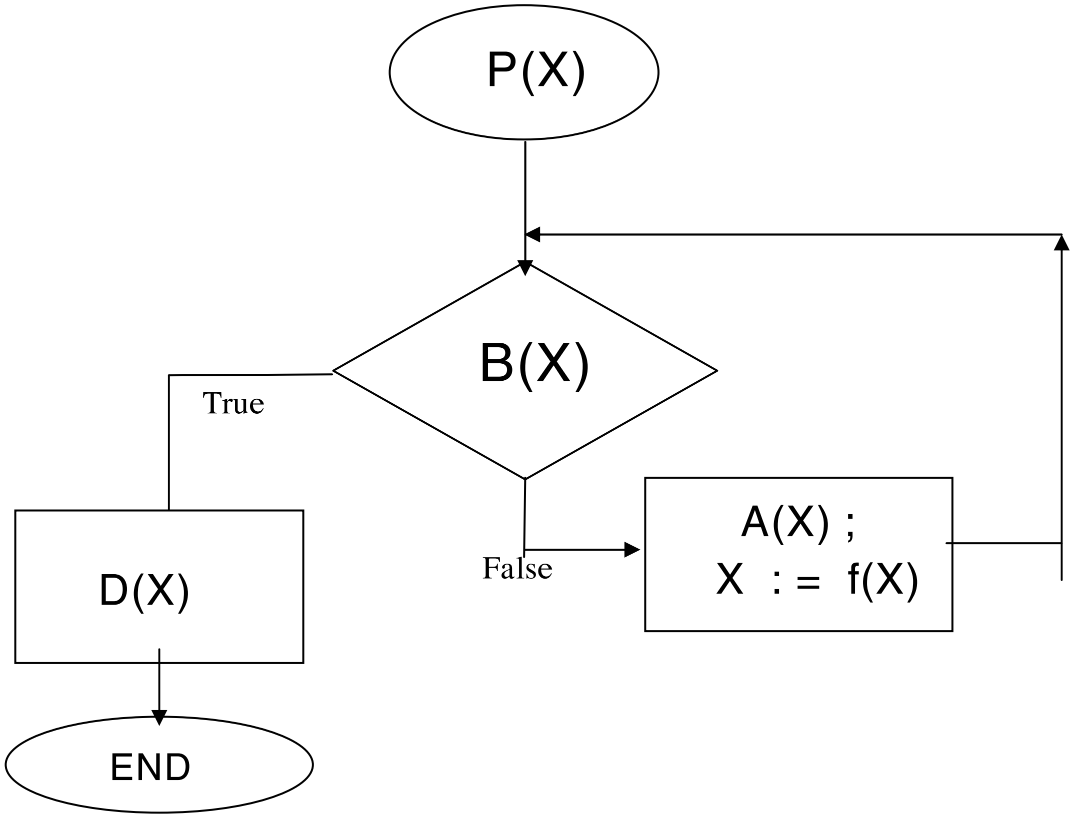
P_i là lần gọi P thứ i (lần i + 1) P(f(f(...f(X)...)

($P(f_i(X))$ hợp i lần hàm f)

Trong lần gọi P_i nếu $B(f_i(X))$ không đúng (false) thì thi hành lệnh A và gọi P_{i+1} ; nếu $B(f_i(X))$ đúng (true) thì thi hành lệnh D và kết thúc qúa trình gọi .

Giả sử P được gọi đúng n +1 lần . Khi đó ở trong lần gọi cuối cùng (thứ n) P_n thì $B(f_n(X))$ đúng , lệnh D được thi hành và chấm dứt thao tác gọi thủ tục P .

Sơ đồ khối quá trình thực hiện lệnh gọi thủ tục P(X) có dạng sau :



Tương ứng với vòng lặp sau :

```
While  ( not B(X) )   do   begin
                              A(X)  ;
                              X := f(X) ;
                           end ;
D(X) ;
```

Ví dụ 1 :
Để đổi 1 số nguyên không âm y ở cơ số 10 sang dạng cơ số k (2 <= k <= 9) với việc dùng mảng A (A : array[1 . . size] of 0..k -1 , size là một hằng được khai báo trước) để chứa các ký số trong hệ k phân (với quy ước ký số có ý nghĩa thấp được chứa ở chỉ số cao) khi đó thủ tục đệ quy Convert(x,m) để tạo dãy gía trị : A[0] , A[1] , . . . , A[m] như sau (hãy tự giải thích) :

```
Convert(n,m)  ≡   if  n <> 0  then   Begin
                                        A[m]  :=  n  mod  k ;
                                        Convert(n  div  k , m -1)  ;
                                     End ;
```

Lệnh gọi Convert(y,n) dùng để đổi số nguyên y trong cơ số 10 sang cơ số k lưu dãy
ký số trong mảng A ;

Trong ví dụ này ta có :

X là (n, m) ;
B(X) là biểu thức boolean not(n <> 0)
A(X) là lệnh gán A[m] := n mod k ;
f(X) là hàm f(n,m) = (n div k , m - 1) ;
D(X) là lệnh rỗng

Đoan lệnh lặp tương ứng với thủ tục Convert(x,m) là :

```
While (n <> 0) then begin
                    A[m] := n mod k ; { A(X) }
                      n := n div k ;  { X := f(X) }
                      m := m - 1 ;
                   end ;
```

Ví dụ 2 :

Tìm USCLN của 2 số nguyên dựa vào thuật toán Euclide .

- Giải thuật đệ quy (dưới dạng thủ tục) tìm USCLN(m,n) bằng thuật toán Euclide :

```
USCLN(m , n , var us)  ≡   if ( n = 0 ) then us := m
                                  else USCLN(n , m mod n , us ) ;
```

- Trong trường hợp này thì :

X là (m , n , us)
P(X) là USCLN(m ,n ,us)
B(X) là n = 0
D(X) là lệnh gán us := m
A(X) là lệnh rỗng
f(X) là f(m,n,us) = (n , m mod n ,us)

- Đoạn lệnh lặp tương ứng là :

```
While (n <> 0 ) do begin
                    sd := m mod n ;
                    m := n ;
                    n := sd ;
                   end ;
us := m ;
```

- Thủ tục không đệ quy tương ứng trong Pascal .

```
Procedure   USCLN(m , n : integer ;  var  us : integer ) ;
     var   sd : integer ;
     begin
          while ( n <> 0 )  do  begin
                                    sd  :=  m  mod  n ;
                                    m  :=  n  ;
                                    n   :=  sd  ;
                                 end ;
          us  :=  m  ;
       end ;
```

- Hàm con không đệ quy tương ứng trong C++

```
void    USCLN(int m , int n , int& us )
     { while(n != 0 )   {  int  sd  = m % n ;
                            m   =   n  ;
                            n    =   sd ;
                         }
       us  =  m  ;
      }
```

c) Các hàm đệ qui dạng đệ qui đuôi (tail-recusive).

Xét hàm đệ qui dạng :

$$f(X) = \begin{cases} f(g(X)) & \text{khi } C(X) \text{ đúng} \\ a(X) & \text{khi } C(X) \text{ sai} \end{cases}$$

Tức là :

$f(X) \equiv$ if(C(X)) then return(f(g(X))
else return(a(x))

Tính $f(X_o)$.

Ta có :

$f(X_o) = f(g(X_o))$ với $C(X_o)$ đúng .

$= f(g(g(X_o)))$ với $C(g(X_o))$ đúng .

$= \ldots$

$= f(g^k(X_o))$ với $C(g^{k-1}(X_o))$ đúng .

$= a(g^k(X_o))$ với $C(g^k(X_o))$ sai.

($g^k(xo) = g(g(g \quad (x_o)))))$)

Đặt : $U_o = X_o = g^o(X_o)$

$U_i = g^i(X_o) = g(g^{i-1}(X_o)) = g(U_{i-1})$ với $i >= 1$

Ta có quan hệ sau :

$U_o = X_o$

$U_i = g(U_{i-1})$ i = 1 ... k . Với k là số nhỏ nhất mà $C(U_k)$ sai .

Lúc đó : $f(X_o) = a(U_k)$

Vậy đoạn chương trình tính f = $f(X_o)$ là :

```
U := Xo ;
while C(U) do U := g(U) ;
f := a(U) ;
```

Ví dụ :

Với m , n > = 0 ta có hàm đệ quy tính USCLN(m,n) là :

USCLN(m ,n) ≡ if (m <> 0) then return(USCLN (abs(m - n) , min(m , n)) ;
else return n ;

Trong trường hợp này :

X là (m ,n) ;

C (X) = C(m ,n) là m <> 0 ;

g(X) = g(m ,n) = (abs(m -n) , min (m ,n)) ;

a(x) = a(m ,n) = n ;

- Đoạn chương trình tính USCLN(a ,b) là :

```
m := a ;  n := b ;
while ( m <> 0 ) do begin
                        t1 := m ;
                        t2 := n ;
                        m := abs(t1 - t2 ) ;
                        n := min(t1,t2 ) ;
                     end ;
USCLN := n ;
```

- Hàm không đệ qui tương ứng trong Pascal.

```
Function USCLN(m , n : integer ) : integer ;
  var t1 , t2 : integer ;
  begin
     while (n <> 0 ) do begin t1 := m ; t2 := n ;
                   m := abs(t1 - t2 ) ;
                   if(t1 < t2 ) then n := t1
                                else n := t2 ;
              end ;
     USCLN := m ;
```

- Dạng hàm tương ứng trong C++

```
int USCLN(int m , int n)
  { while( n != 0) { int t1 = m ; int t2 = n ;
```

```
                m   =  abs(t1-t2) ;
                if(t1<t2)  n = t1  ; else  n = t2 ;
              }
        return(m) ;
        }
```

2. Khử đệ quy hàm đệ quy arsac

a) Dạng hàm đệ qui ARSAC.

a1) Dạng toán học :

$$A(X) = \begin{cases} DS(A(CS(X)), FS(CS(X), X)) & \text{khi } C(X) \text{ đúng} \\ BS(X) & \text{khi } C(X) \text{ sai} \end{cases}$$

a2) Dạng mã giả :

```
A(X ) ≡  if C(X)  then  return ( DS (A(CS(X)) ,FS(CS(X),X) )
                        else   return (BS(X ) )
```

Với : BS , CS , DS , FS là các giải thuật không đệ qui .

Trường hợp thường gặp là : BS(X) , CS(Y) , DS(U,V) , FS(U,V) là các thao tác đơn giản , không có lệnh gọi hàm con . X , Y ,U , V là biến đơn trị hoặc biến véc tơ .

Đây là dạng tổng quát của hàm đệ quy chỉ gọi đến chính nó một lần .

b) Sơ đồ tổng quát tính gía trị A(X) :

Gọi $U_o = X$ là gía trị đối số cần tính của hàm A . Việc tính $A(U_o)$ sẽ phát sinh lệnh gọi tính $A(U_1)$ với $U_1 = CS(U_o)$ (gỉa sử $C(U_o)$ true).

Cứ như vậy , khi mà $C(U_i)$ còn đúng thì việc tính $A(U_i)$ sẽ phát sinh lệnh tính $A(U_{i+1})$ với $U_{i+1} = CS(U_i)$.

Với giả thiết là $U_o = X$ thuộc miền xác định của A , thì quá trình lặp lại các lệnh gọi này phải dừng lại sau hữu hạn lần gọi . Tức là $\exists$ k thỏa :

$$C(U_o) = C(U_1) = . . = C(U_{k-1}) = \text{true} , C(U_k) = \text{false}.$$

Xét 2 dãy số :

- Dãy : $\{ U_i \} = \{ CS(U_i) \}$ (2.1)

  $U_o = X$ { cho trước }

  $U_{i+1} = CS(U_i)$ $i = 0 .. k-1$

- Dãy : $\{ V_i \} = \{ A(U_i) \}$ (2.2)

  $V_o = A(U_o) = A(X_o)$ (gía trị cần tính).

  $V_i = A(U_i) = DS(A(CS(U_i)), FS(CS(U_i), U_i))$

$= DS(A(U_{i+1}),FS(U_{i+1},U_i))$

$= DS(V_{i+1},FS(U_{i+1},U_i))$ với $0 < i < k$ (vì $C(U_i)$ đúng)

$V_k = BS(U_k)$ (vì $C(U_k) = false$)

Dựa vào 2 dãy số $\{U_i\}$,$\{V_i\}$ (mô tả bởi (2.1) và (2.2)) ta tính A(X) theo giải thuật sau :

- Tính và ghi nhớ các U_i từ 0 đến k theo (2.1).

(Với $C(U_o) = C(U_1) = ...= C(U_{k-1}) = True$, $C(U_k) = False$)

- Sử dụng dãy gía trị U_i để tính lần ngược V_i từ k xuống 0 theo (2.2) , V_o chính là gía trị cần tính ($V_o = A(X)$).

c) Giải thuật không đệ quy tính gía trị hàm Arsac bằng sử dụng cấu trúc Stack .

Để thực hiện giải thuật trên thì dãy U_i phải được tính và lưu trữ trong một cấu trúc dữ liệu thích hợp , để khi cần đến (khi tính V_i) dễ lấy ra sử dụng . Đặc điểm quan trong của dãy U_i là thỏa luật LIFO : thứ tự sử dụng ngược với thứ tự tạo sinh . Cấu trúc dữ liệu cho phép lưu trữ thuận lợi dãy phần tử thỏa luật LIFO (vào sau ra trước - Last In First Out) là câu trúc Stack .

(Trên cấu trúc Stack 2 thao tác cơ bản đặc trưng là :

+ Push(S,X) : Chèn phần tử dữ liệu X vào đỉnh Stack S .

+ Pop(S,X) : Lấy ra khỏi stack S phần tử dữ liệu ở đỉnh và chứa nó vào biến X).

Giải thuật không đệ qui tính $V_o = A(X)$ dựa trên 2 công thức (2.1) , (2. 2) và sử dụng Stack S là :

+ Bước 1 : tính U_i bắt đầu từ U_o theo (2.1) lưu vào Stack S

```
CreateStack(S) ;   ( tạo stack rỗng S )
k := 0 ;
U := X  ; ( Uo = X )
push(S,U) ; ( chèn UO vào đỉnh stack S )
while  C(U)  do  begin
                    k := k+1 ;
                    U := CS(U) ; ( Uk+1 = CS(Uk))
                    push (S,U) ; ( chèn Uk+1 vào đỉnh Stack S )
                  end ;
```

+ Bước 2 : Lấy dữ liệu trong Stack S tính V_i theo (2. 2)

```
pop(S,U)  ; ( U = Uk )
V := BS(U)  ;    ( C(Uk) sai ; V=Vk = BS (Uk))
for  i  :=  k -1   downto  0   do
     begin
          Y  :=  U  ;                  ( Y  =  Ui+1)
```

pop(S,U) ; ($U = U_i$)
V := DS(V,FS(Y,U)) ; ($C(U_i)$ đúng ; $V_i = DS(V_{i+1},FS(U_{i+1},U_i))$)
end ;
{ $V = A(X_o)$ }

Hoặc :

+ Bước 1 : tính U_i bắt đầu từ U_o theo (2.1) lưu vào Stack S
CreateStack(S) ; (tạo stack rỗng S)
U := X_o ; ($U_o = X_o$)
push(S,U) ; (chèn U_O vào đĩnh stack S)
while C(U) do begin
U := CS(U) ; ($U_{K+1} = CS(U_K)$)
push (S,U) ; (chèn U_{k+1} vào đĩnh Stack S)
end ;

+ Bước 2 : Lấy dữ liệu trong Stack S tính V_i theo (2. 2)
pop(S,U) ; ($U = U_k$)
V := BS(U) ; ($C(U_k)$ sai ; $V=V_k = BS(U_k)$)
While(not emptystack(S)) do
begin
Y := U ; ($Y = U_{i+1}$)
pop(S,U) ; ($U = U_i$)
V := DS(V,FS(Y,U)) ; ($C(U_i)$ đúng ; $V_i = DS(V_{i+1},FS(U_{i+1},U_i))$)
end ;
{ $V = A(X_o)$ }

Cơ chế lưu trử dãy dữ liệu LIFO bằng Stack là một đặc trưng của quá trình xử lý giải thuật đệ quy điều cần quan tâm là cấu trúc stack thường chiếm nhiều bộ nhớ . Vì vậy người ta luôn tìm cách tránh dùng nó khi còn tránh được .

d) Một số hàm Arsac đặc biệt mà việc khử đệ qui giải thuật tính gía trị hàm có thể không dùng Stack .

d1) Trường hợp thủ tục CS là song ánh .

Trường hợp CS là song ánh từ miền D lên miền D thì hàm CS có hàm ngược CS^{-1} . Gọi hàm ngược của hàm CS là hàm CSM1 .

Ta có : $CSM1(CS(X)) = CS^{-1}(CS(X)) = X$ với $\forall X \in D$.

Nên : $CSM1(U_{i+1}) = CS^{-1}(CS(U_i)) = U_i$ với $i = k-1, . . ,1,0$

Khi đó ta không cần lưu giữ các giá trị trung gian của dãy { Ui } mà chỉ cần xuất phát từ U_k dùng hàm CSM1 để khôi phục lại các gía trị U_i với i<k .

Giải thuật tính A(X) sẽ trở thành :

+ Bước 1 : Dựa vào (2.1) tính U_k

```
U := X ; ( Uo = X )
k := 0 ;
while C(U) do begin
        k := k+1 ;
        U := CS(U) ; ( UK+1 = CS(UK))
      end ;
```

+ Bước 2 : Tính Vk , Vk-1, .. V1, Vo dựa vào Uk ,(2.2) và CSM1

```
V := BS(U) ; ( V=Vk = BS (Uk) )
for i := k -1 downto 0 do begin
        Y := U ;      ( Y = Ui+1 )
        U := CSM1(U) ; (Ui = CSM1(Ui+1) )
        V := DS(V,FS(Y,U)) ;
           ( Vi = DS(Vi+1,FS(Ui+1,Ui) )
      end ;
{ V = Vo = A(X )}
```

d2) Trường hợp thủ tục DS có tính hoán vị .

Xét công thức tính :

$V_i = DS(V_{i+1},FS(U_{i+1},U_i))$ với mọi $i<k$

Đặt : $U'_i = FS(U_{i+1},U_i)$

$DS(V_{i+1},U'_i) = V_{i+1}\ T\ U'_i$

Ta có :

$V_o = DS(V_1, FS(U_1,U_o)) = DS(V_1,U'_o) = V_1\ T\ U'_0$

$V_1 = DS(V_2, FS(U_2,U_1)) = DS(V_2,U'_1) = V_2\ T\ U'_1$

$V_2 = DS(V_3, FS(U_3,U_2)) = DS(V_3,U'_2) = V_3\ T\ U'_2$

....

....

$V_i = DS(V_{i+1}, FS(U_{i+1},U_i)) = DS(V_{i+1},U'_i) = V_{i+1}\ T\ U'_i$ (3 - 1)

....

....

$V_{k-1} = DS(V_k, FS(U_k,U_{k-1})) = DS(V_k,U'_{k-1}) = V_k\ T\ U'_{k-1}$

$V_k = BS(U_k)$

Khi DS có tính hoán vị tức : DS(DS(x,y),z) = DS(DS(x,z),y)

(Viết trên ký hiệu T : (x T y) T z = (x T z) T y

Thực hiện thế lần lượt V_1 rồi V_2 ... trong công thức V_o.

Ta có :

$$V_o = V_1 \,T\, U'_0 = (V_2 \,T\, U'_1) \,T\, U_o = (V_2 \,T\, U'_0) \,T\, U'_1$$
$$= ((V_3 \,T\, U'_2) \,T\, U'_o) \,T\, U'_1 = ((V_3 \,T\, U'_2) \,T\, U'_o) \,T\, U'_1$$
$$= ((V_3 \,T\, U'_o) \,T\, U'_2) \,T\, U'_1$$
$$= ((V_3 \,T\, U'_o) \,T\, U'_1) \,T\, U'_2$$

....

....

$$V_0 = (\ldots (((V_k \,T\, U'_o) \,T\, U'_1) \,T\, U'_2) \,T\, \ldots\ldots T\, U'_{k-2}) \,T\, U'_{k-1} \quad (3 - 2)$$

(3 - 2) là một dãy liên tiếp (một tổng) k phép toán T mà ta đã biết giải thuật tính. Thực vậy :

Thiết lập dãy Wi như sau :

$W_0 = V_k$

$W_i = W_{i-1} \,T\, U'_{i-1}$ với $i = 1..k$

Tức là : $W_o = V_k = BS(U_k)$ (3 - 3)

$W_i = W_{i-1} \,T\, U'_{i-1} = DS(W_{i-1}, FS(U_i, U_{i-1}))$ $i=1..k$

W_k chính là gía trị V_o cần tính .

Như vậy giải thuật tính W_k (Vo = A(X)) gồm 2 bước :

Bước 1: Xác định k và U_k theo công thức (1 - 1)

Bước 2: Tính dãy W_i , trong lúc tính thì phải tính lại dãy U_i ,theo (3 - 3)

A(X) = Vo = Wk .

Giải thuật không đệ qui tương ứng dược xem như bài tập .

3. Khử đệ quy một số dạng thủ tục đệ quy thường gặp.

a) Dẫn nhập.

Để thực hiện một chương trình con đệ quy thì hệ thống phải tổ chức vùng lưu trữ thỏa quy tắc LIFO (vùng Stack). Vì vậy chỉ những ngôn ngữ lập trình có khả năng tạo vùng nhớ stack mới cho phép tổ chức các chương trình con đệ quy. Thực hiện một chương trình con đệ quy theo cách mặc định thường tốn bộ nhớ vì cách tổ chức Stack một cách mặc định thích hợp cho mọi trường hợp thường là không tối ưu trong từng trường hợp cụ thể. Vì vậy sẻ rất có ích khi người lập trình chủ động tạo ra cấu trúc dữ liệu stack đặc dụng cho từng chương trình con đệ quy cụ thể .

Phần tiếp theo sẻ trình bày việc khử đệ quy một số dạng thủ tục đệ quy theo hướng thay giải thuật đệ quy bằng các vòng lặp và một cấu trúc dữ liệu kiểu stack thích hợp .

b) Thủ tục đệ qui chi có một lệnh gọi đệ quy trực tiếp .

Mô hình tổng quát của thủ tục đệ quy chỉ có một lệnh gọi đệ quy trực tiếp là :

P(X) ≡ if C(X) then D(X)

else begin

A(X) ; P(f(X)) ; B(X) ;

end ;

Với :

X là một biến đơn hoặc biến véc tơ.

C(X) là một biểu thức boolean của X .

A(X) , B(X) , D(X) là các nhóm lệnh không đệ quy (không chứa lệnh gọi đến P).

f(X) là hàm của X .

Tiến trình thực hiện thủ tục P(X) sẻ là :

+ Nếu C(X) đúng thì thực hiện D(X) .

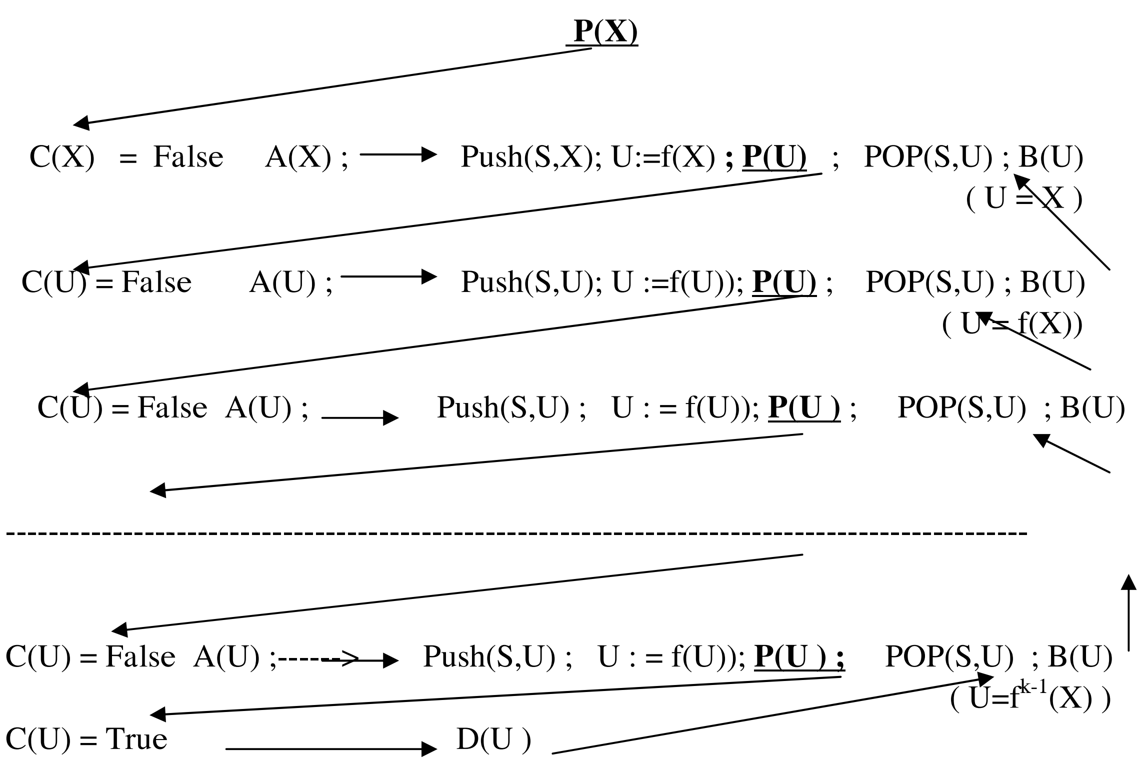
+ Còn không (C(X) sai) thì thực hiện A(X) ; gọi P(f(X)) ; thực hiện B(X) . (B(X) chỉ được thực hiện khi P(f(X)) thực hiện xong) .

Mỗi lần thành phần đệ quy P(Y) được gọi thì thông tin giải thuật B(Y) lại được sinh ra (nhưng chưa thực hiện) .

Gỉa sử qúa trình đệ quy kết thúc sau k lần gọi đệ quy thì ta phải thực hiện một dãy k thao tác B theo thứ tự : $B(f^{k-1}(X))$, $B(f^{k-2}(X))$, . . . ,B(f(f(X))) ,B(f(X)),B(X).

Để thực hiện dãy thao tác { $B(f^{i}(X))$ } theo thứ tự ngược với thứ tự phát sinh ta cần dãy dữ liệu {$f^{i}(X)$ } truy xuất theo nguyên tắc LIFO. Ta sẻ dùng một Stack để lưu trử dãy { $f^{i}(X)$ } ≡ { X , f(X) , f(f(X)) , . . . , $f^{i}(X)$, . . . , $f^{k-1}(X)$ }

Trình tự thực hiện P(X) được diễn tả bằng mô hình sau :



Giải thuật thực hiện P(X) với việc sử dụng Stack có dạng :

```
P(X)    ≡  {   Creat_Stack (S)  ;  ( tạo stack S )
              While(not(C(X))    do  begin  A(X) ;
                                           Push(S,X) ; ( cất gía trị X vào stack S )
                                            X  :=   f(X)  ;
                                          end ;
               D(X)  ;
              While(not(EmptyS(S)))  do  begin
                                              POP(S,X) ; ( lấy dữ liệu từ  S )
                                              B(X)  ;
                                            end ;
               }
```

Ví dụ :

Thủ tục đệ quy chuyển biểu diễn số từ cơ số thập phân sang nhị phân có dạng :

```
Binary(m)    ≡   if ( m > 0 ) then  begin
                                     Binary( m  div  2 ) ;
                                     write( m  mod  2 ) ;
                                   end;
```

Trong trường hợp này :

- X là m .
- P(X) là Binary(m) .
- A(X) ; D(X) là lệnh rỗng .
- B(X) là lệnh Write(m mod 2) ;
- C(X) là (m <= 0) .
- f(X) = f(m) = m div 2 .

Giải thuật thực hiện Binary(m) không đệ quy là :

```
Binary (m )  ≡    {   Creat_Stack (S)  ;
                    While ( m > 0 )   do    begin
                                               sdu  := m  mod  2  ;
                                               Push(S,sdu) ;
                                               m := m  div  2  ;
                                             end;
                  While( not(EmptyS(S))  do  begin
                                                 POP(S,sdu) ;
                                                 Write(sdu) ;
                                               end;
                   }
```

c) Nhiều lệnh gọi đệ quy trực tiếp.

c1) Thủ tục đệ quy với 2 lần gọi trực tiếp

Thủ tục đệ quy 2 lần gọi trực tiếp có dạng :

```
P(X) ≡   if  C(X)  then   D(X)
                  else  begin
                          A(X) ;  P(f(X)) ;
                          B(X) ;  P(g(X)) ;
                        end ;
```

Qúa trình thực hiện thủ tục P(X) sẻ là :

- Nếu C(X) đúng thì thực hiện D(X) .
- Nếu C(X) sai thì thực hiện A(X) ; gọi P(f(X)) ; thực hiện B(X) ; gọi P(g(X)) , khi gọi P(g(X)) thì lại phát sinh lệnh A(g(X)) như vậy ngoài việc phải lưu vào stack các gía trị $f^i(X)$ ta con phải lưu vào stack các gía trị $g^i(X)$ tương ứng . Khi ta lấy dữ liệu từ stack để thực hiện lệnh B(U) mà chưa gặp điều kiện kết thúc thì ta thực hiện P(g(U)) và lại phải lưu gía trị g(U) vào stack ,... Điều kiện dừng là khi truy xuất tới phần tử lưu đầu tiên trong stack .

Như vậy là ngoài dữ liệu X , con phải lưu vào ngăn xếp thêm thứ tự lần gọi (cụm gọi)

Thuật toán khử đệ quy tương ứng với thủ tục đệ quy P(X) là :

```
{ Creat_Stact (S) :
  Push (S, (X,1)) ;
  Repeat
     While ( not C(X) ) do  begin
                              A(X) ;
                              Push (S, (X,2)) ;
                              X := f(X) ;
                            end ;
      D(X) ;
      POP (S, (X,k)) ;
      if ( k <> 1) then  begin
                           B(X) ;
                           X := g(X) ;
                         end ;
   until ( k = 1 ) ;
 }
```

Ví dụ : Khử đệ quy thủ tục Tháp Hà Nội .

+ Dạng đệ quy của thủ tục Tháp Hà Nội là :

```
THN(n , X , Y, Z )  ≡   if( n > 0 )  then  begin
                                              THN ( n - 1 , X , Z , Y ) ;
                                               Move ( X , Z ) ;
                                               THN ( n - 1 , Y , X , Z ) ;
                                             end ;
```

Với n là số đĩa , X là cột đầu , Z là cột cuối , Y là cột giữa ,Move(X,Z) là thao tác chuyển 1 đĩa từ cột X tới cột Z .

Trong trường hợp này :

Biến X là bộ (n , X , Y , Z) .
C(X) là biểu thức boolean (n < = 0) .
D(X) , A(X) là thao tác rỗng .
B(X) = B(n,X,Y,Z) là thao tác Move(X,Z) ;
f(X) = f(n ,X ,Y ,Z) = (n - 1 , X , Z , Y) .
g(X) = g(n ,X , Y, Z) = (n - 1 , Y ,X , Z) .

Giải thuật không đệ quy tương đương là :

```
{  Creat_Stack (S)  ;
   Push (S ,(n,X,Y,Z,1))  ;
   Repeat
      While ( n > 0 )  do   begin
                              Push (S ,(n,X,Y,Z,2)) ;
                               n  :=  n - 1 ;
                               Swap (Y,Z ) ;  (* Swap(a,b) là thủ tục hoán
                         end ;                   đổi nội dung 2  biến a ,b   *)
      POP (S,(n,X,Y,Z,k)) ;
      if ( k <> 1 )   then   begin
                               Move (X ,Z ) ;
                               n  :=  n - 1 ;
                               Swap (X ,Y ) ;
                             end ;
   until ( k = 1 ) ;
  }
```

c2) Trường hợp n lần gọi đệ quy trực tiếp .

Thủ tục đệ quy trong trường hợp này có dạng :

```
P(X)  ≡   if  C(X)  then    D(X)
                  else   begin
                            A1(X)  ;   P(f1(X)) ;
                            A2(X)  ;   P(f2(X)) ;
                              ..........................
                           Ai(X)  ;   P(fi(X)) ;
                              ..........................
                           An(X)  ;   P(fn(X)) ;
                           An+1(X) ;
                         end ;
```

Cũng giống như trong trường hợp (3a) là khi quay trở lại sau khi thực hiện một lần đệ quy, cần biết đó là lệnh gọi thuộc nhóm thứ mấy trong dãy lệnh gọi để biết thao tác cần thực hiện tiếp. Vì vậy trong chồng cần giữ thêm vị trí nhóm lệnh gọi .

Dạng lặp tương ứng là :

```
{ Creat_Stack (S) ;
  Push(S,(X,1)) ;
  Repeat
     While (not C(X) )  do  begin
                              A1(X)  ;
                              Push (S,(X,2))  ;
                              X  :=  f1(X) ;
                            end ;
     D(X)  ;
      POP(S,(X,k)) ;
            While( k = n+1 )  do  begin
                              An+1 ;
                              POP(S,(X,k))  ;
                            end ;
      if ( k > 0 )  then    begin
                              Ak(X) ;
                            Push (S,(X,k+1));
                            X  :=  f k (X)
                          end ;
    until (k = 1 ) ;
   }
```

Ví dụ : Khử đệ quy cho thủ tục hoán vị .

+ Thủ tục hoán vị dưới dạng đệ quy :

```
HVI(V ,n)   ≡   if (n = 1 )  then   Print ( V )
                                else  for  i  :=  n  downto  1  do
                                         begin
                                            Swap (V[n],V[i] ) ;
                                             HVI(V ,n - 1) :
                                         end ;
```

trong trường hợp này thì :

X là bộ (V ,n) . (* vector V và số nguyên n *)

C(X) là (n = 1) .

D(X) là Print (V) . (* xuất vector V *)

Ai(X) là thủ tục Swap(V[n] ,V[i]) (i = 1 .. n) .

An+1 là thao tác rỗng .

fi(X) = f(V, n) = (V, n - 1) .(với i = 1 . . n)

Dạng lặp của thủ tục là :

```
{ Creat_Stack (S) ;
  Push (S,(V ,n ,1))  ;
  Repeat
     While ( n > 1 )  do   begin
                              Swap(V[n] ,V[1] ;
                              Push (S ,V , n ,2) ;
                               n  :=  n -1 ;
                             end ;
       Print (V) ;
       POP (S ,(V ,n ,k)) ;
     While ( k = n +1 )  do   POP(S ,(V ,n ,k) ;
       if(k <> 1 ) then  begin
                            Swap(V[n] ,V[k]) ;
                             Push (S ,(V ,n ,k+1) ;
                              n := n - 1 ;
                          end ;
    until(k = 1 ) ;
```

PHẦN II

KIỂM CHỨNG CHƯƠNG TRÌNH

CHƯƠNG IV
CÁC KHÁI NIỆM

I. CÁC GIAI ĐOẠN TRONG CUỘC SỐNG CỦA MỘT PHẦN MỀM

Việc sử dụng máy tính để giải một bài toán thực tế thường bao gồm nhiều việc. Trong các công việc đó công việc mà người ta quan tâm nhất là việc xây dựng các hệ thống phần mềm (các hệ thống chương trình giải bài toán).

Để xây dựng một hệ thống phần mềm , người ta thường thực hiện trình tự các công việc sau : Đặc tả bài toán, xây dựng hệ thống, sử dụng và bảo trì.

1) Đặc tả bài toán

Gồm việc phân tích để nắm bắt rõ yêu cầu của bài toán và diễn đạt chính xác lại bài toán bằng ngôn ngữ thích hợp vừa thích ứng với chuyên ngành tin học vừa có tính đại chúng (dễ hiểu đối với nhiều người).

2) Xây dựng hệ thống

Trong bước này sẻ tuần tự thực hiện các công việc sau :

- Thiết kế : Xây dựng mô hình hệ thống phần mềm cần có. Trong bước này, công việc chủ yếu là phân chia hệ thống thành các module chức năng và xác định rõ chức năng của từng module cũng như mối tương tác giữa các module với nhau. Chức năng của mỗi module được định rõ bởi đặc tả của từng module tương ứng.

- Triển khai từng module và thử nghiệm :

Viết chương trình cho từng module (bài toán con) thỏa "đúng" đặc tả đã đặt ra. Tính đúng của chương trính được quan tâm bằng 2 hướng khác nhau :

+ Chứng minh tính đúng một cách hình thức (thường là một công việc khó khăn) .

+ Chạy thử chương trình trên nhiều bộ dữ liệu thử khác nhau mỗi bộ dữ liệu đại diện cho một lớp dữ liệu (thường là một công việc tốn kém). Để có tính thuyết phục cao, người ta cần chạy thử trên càng nhiều bộ dữ liệu càng tốt. Khi thử nếu phát hiện sai thì phải sửa lại chương trình còn chưa phát hiện sai thì ta con tạm tin chương trình đúng (chạy thử chỉ có tác dụng phát hiện sai và tăng lòng tin vào tính đúng chứ không chứng minh được tính đúng).

- Thử nghiệm ở mức độ hệ thống : Sau khi từng module hoạt động tốt, người ta cần thử sự hoạt động phối hợp của nhiều module, thư nghiệm toàn bộ hệ thống phần mềm.

Thử nghiệm tính đúng theo bất cứ cách nào thì cũng rất tốn thời gian và công sức nhưng lại là một việc phải làm của người lập trình vì người lập trình luôn luôn phải bảo đảm chương trình mình tạo ra thỏa đúng đặc tả.

3) Sử dụng và bảo trì hệ thống

Sau khi hệ thống phần mềm hoạt động ổn định, người ta đưa nó vào sử dụng.

Trong quá trình sử dụng có thể có những điều chỉnh trong đặc tả của bài toán, hay phát hiện lỗi sai của chương trình. Khi đó cần xem lại chương trình và sửa đổi chúng.

Các yêu cầu sau cho qúa trình xây dựng phần mềm :

a) Cần xây dựng các chương trình dễ đọc, dễ hiểu và dễ sửa đổi.

Điều này đòi hỏi một phương pháp tốt khi xây dựng hệ phần mềm : phân rã tốt hệ thống , sử dụng các cấu trúc chuẩn và có hệ thống khi viết chương trình ,có sưu liệu đầy đủ .

b) Cần đảm bảo tính đúng. Làm thế nào để xây dựng một chương trình "đúng" ?

Một điều cần chú ý là: Phép thử chương trình chỉ cho khả năng chỉ ra chương trình sai nếu tình cờ phát hiện được chứ không chứng minh được chương trình đúng vì không thể thử hết được mọi trường hợp. Vì vậy người ta luôn cố gắng chứng minh chương trình đúng của chương trình bằng logic song song với chạy thử chương trình.

Có 2 cách chính thường được sử dụng để đảm bảo tính đúng của phần mềm trong quá trình xây dựng hệ thống :

- Viết chương trình rồi chứng minh chương trình đúng.
- Vừa xây dựng vừa chứng minh tính đúng của hệ thống.

Việc tìm kiếm những phương pháp xây dựng tốt để có thể vừa xây dựng vừa kiểm chứng được tính đúng luôn là một chủ đề suy nghĩ của những người lập trình .

II. ĐẶC TẢ

1. Đặc tả bài toán

a) Khái niệm.

Khi có một vấn đề (một bài toán) cần được giải quyết , người ta phát biểu bài toán bằng một văn bản gọi là đặc tả bài toán (problem specification).

Các bài toán đặt ra cho những người làm công tác tin học thường có dạng sau : Xây dựng một hệ thống xử lý thông tin mà hoạt động của nó :

- Dựa trên tập dữ liệu nhập (thông tin vào) thoả mãn những điều kiện nhất định.
- Xẩy ra trong một khung cảnh môi trường hạn chế nhất định.
- Mong muốn sản sinh ra một tập dữ liệu xuất (thông tin ra) được quy định trước về cấu trúc và có mối quan hệ với dữ liệu nhập và môi trường được xác định trước .

Những khía cạnh trên được thể hiện trong đặc tả bài toán (ĐTBT) .

b) Tác dụng của đặc tả bài toán .

- Là cơ sở để đặt vấn đề, để truyền thông giữa những người đặt bài toán và những người giải bài toán .
- Là cơ sở để những người giải bài toán triển khai các giải pháp của mình .
- Là cơ sở để những người giải bài toán kiểm chứng tính đúng của phần mềm tạo ra .
- Là phương tiện để nhiều người hiểu tính năng của hệ thống tin học mà không cần (thường là không có khả năng) đi vào chi tiết của hệ thống .

Để đạt được 4 mục tiêu trên, ĐTBT cần gọn, rõ và chính xác .

Để đạt được mục tiêu thứ 2, thứ 3 thì ngôn ngữ để viết ĐTBT cần phải có tính hình thức (formal) và trên ngôn ngữ này cần có các phương tiện để thực hiện các chứng minh hình thức . Ngôn ngữ thích hợp với yêu cầu này là ngôn ngữ toán học và hệ logic thích hợp là logic toán học. Người ta thường sử dụng ngôn ngữ bậc nhất (với các khái niệm và toán tử toán học) và logic bậc nhất .

Tuỳ theo mức độ phức tạp của bài toán mà phương tiện diễn đạt ĐTBT có những mức độ phức tạp và mức độ hình thức khác nhau .

Ở mức bài toán lớn, trong mối quan hệ giữa người sử dụng và người phân tích, người ta dùng : sách hợp đồng trách nhiệm (cahier des charges), sơ đồ tổ chức, biểu đồ luân chuyển thông tin ... Giữa những người phân tích, người ta dùng phiếu phân tích các đơn vị chức năng, biểu đồ chức năng...

Kết quả phân tích được chuyển thành yêu cầu với người lập trình bằng các đặc tả chương trình (ĐTCT - program specification) .

2. Đặc tả chương trình (ĐTCT).

ĐTCT gồm các phần sau :

- Dữ liệu nhập : Các dữ kiện mà chương trình sử dụng . Đặc trưng quan trọng là danh sách dữ liệu nhập và cấu trúc của chúng , có khi cần nêu nguồn gốc , phương tiện nhập của mỗi dữ liệu nhập .
- Điều kiện ràng buộc trên dữ liệu nhập : là những điều kiện mà dữ liệu nhập phải thoả để hệ thống hoạt động đúng . Chương trình không bảo đảm cho kết quả đúng khi thực thi các bộ dữ liệu không thoả các điều kiện này .

Trong phần mô tả dữ kiện nhập cần nêu lên :

+ Cấu trúc : kiểu dữ liệu (các thành phần, sự kết nối các thành phần).

+ Các ràng buộc trên gía trị của chúng .

- Dữ liệu xuất : Các dữ liệu mà chương trình tạo ra . Cũng như phần dữ liệu nhập, cần nêu rõ danh sách dữ liệu xuất, cấu trúc của chúng, có khi cần nêu phương tiện xuất của từng dữ liệu xuất.

- Điều kiện ràng buộc trên dữ liệu xuất: Những điều kiện ràng buộc mà dữ liệu xuất phải thoả. Chúng thể hiện yêu cầu của người sử dụng đối với chương trình. Các điều kiện này thường liên quan đến dữ liệu nhập .

Ví dụ 1 :

Viết chương trình đọc vào một số nguyên dương N rồi xuất ra màn hình N số nguyên tố đầu tiên.

Đặc tả chương trình :

+ Dữ liệu nhập : một số nguyên N .
+ Điều kiện nhập : N > 0 , nhập vào từ bàn phím.
+ Dữ liệu xuất : một dãy gồm N số nguyên .
+ Điều kiện xuất : là dãy N số nguyên tố đầu tiên , xuất ra màm hình .

Ví dụ 2 :

Viết chương trình đọc vào một dãy N số nguyên , xuất ra màn hình dãy đã sắp xếp theo thứ tự không giảm.

Đặc tả chương trình :

+ Dữ liệu nhập : một array A có N phần tử là số nguyên .
+ Điều kiện nhập : nhập từ bàn phím .
+ Dữ liệu xuất : array A' có N phần tử là số nguyên.
+ Điều kiện xuất : xuất ra màn hình ,A' là một hoán vị của A , A' là một dãy không giảm. (1 <= i < j <= N ==> A'[i] <= A'[j])

Chú ý : Một đặc tả tốt cho một định hương đúng về sử dụng hợp lý các cấu trúc dữ liệu và một gợi ý tốt về hướng xây dựng giải thuật cho bài toán.

3. Đặc tả đoạn chương trình .

a) Không gian trạng thái.

Một chương trình sử dụng một tập các biến xác định. Một biến thuộc một kiểu dữ liệu xác định. Một kiểu dữ liệu xác định một tập gía trị mà mỗi biến thuộc kiểu có thể nhận .

Tập giá trị mà biến chương trình X có thể nhận (miền xác định của biến X) gọi là không gian trạng thái (state space) của biến X .

Xét chương trình P giả sử P sử dụng các biến a , b , c ,. . . với các không gian trạng thái tương ứng là : A , B , C ,... thì tích Decartes của A,B,C,... (A ^ B ^ C ^ ..) là không gian trạng thái của chương trình P .

b) Đặc tả đoạn chương trình.

Xét một tiến trình xử lý thực thi một chương trình . Mỗi lệnh của chương trình biến đổi trạng thái các biến của chương trình từ trạng thái này sang trạng thái khác , xuất phát từ trạng thái đầu (trạng thái khi bắt đầu tiến trình xử lý) kết thúc tại trạng thái cuối (trạng thái khi tiến trình xử lý kết thúc).

Ở từng thời điểm trước hoặc sau khi thực hiện một lệnh , người ta quan tâm đến tập hợp các trạng thái có thể của chương trình. Tập hợp các trạng thái này sẻ được biểu thị bởi các tân từ bậc nhất với các biến là các biến của chương trình.

Ví dụ 1 : Đoạn chương trình sau tính tích của hai số nguyên dương a và b

```
    { ( a > 0) and (b > 0) }      //  ràng buộc trên trạng thái đầu .
        x := a ;
        y := b ; u := 0 ;
  { ( x = a ) and (y = b ) and ( (u + x*y ) = (a*b) ) }  //  ràng buộc trung gian trên
      repeat      {(u+x*y = a*b) and ( y>0 ) }          trạng thái của CT.
         u := u + a ;
         y := y - 1 ;
     {(u+x*y = a*b) and (y >= 0) }                 // ràng buộc trung gian trên trạng
thái
      until (y= 0)                                  của chương trình.
          {u= a*b}                                //  ràng buộc trên trạng thái xuất
```

Mỗi tân từ trong ví dụ trên mô tả một tập các trạng thái có thể có ở điểm đó.

Ví dụ 2 : Đoạn chương trình hoán đổi nội dung của 2 biến x và y, dùng biến t làm trung gian.

{($x = x_o$) and ($y = y_o$) }   // x , y mang giá trị ban đầu bất kỳ nào đó

t := x ;

x := y ;

y := t

{ ($x = y_o$) and ($y = x_o$) } // x , y sau cùng mang giá trị hoán đổi của nhau.

Trong ví dụ này để biểu diễn quan hệ giữa nội dung các biến với nội dung của một số biến bị gán trị, người ta cần phải dùng các biến giả (ghost variable). Ví dụ ở đây là x_o và y_o biểu thị nội dung của x và y trước khi thực hiện đoạn chương trình.

Ví dụ 3 :

Nhân 2 số nguyên dương x , y, kết quả chứa trong u .

{ ($x = x_o > 0$) and ($y = y_o > 0$)}

u := 0 ;

repeat

u := u + x ;

y := y - 1 ;

until (y = 0)

{ $u = x_o * y_o$ }

Thật ra ở đây tập hợp các trạng thái xuất thực sự là nhỏ hơn, biểu thị bởi một điều kiện chặt hơn, đó là : {($u = x_o * y_o$) and (y = 0) and ($x = x_o$) }

Tổng quát ta cần khảo sát một đoạn chương trình S với 2 điều kiện đi trước P và đi sau Q . Cần chứng minh rằng nếu xuất phát từ trạng thái thoả P thi hành lệnh S thì

cần đạt tới trạng thái thỏa Q . P được gọi là điều kiện đầu (precondition) , Q được gọi là điều kiện cuối (postcondition). Cặp tân từ (P,Q) , được gọi đặc tả của đoạn lệnh S. Bộ 3 S, P, Q tạo nên một đặc tả đoạn lệnh thường được mô tả hình thức bằng tập ký hiệu : { P } S { Q }
({ P } : tập trạng thái thỏa tân từ P , { Q } : tập trạng thái thỏa tân từ Q)

Việc thiết lập các khẳng định ở những điểm khác nhau trong chương trình nhằm để :
+ Hoặc là đặc tả một đoạn chương trình cần phải xây dựng : tìm S thỏa P, Q cho trước.
+ Hoặc là cơ sở để chứng minh tính đúng của đoạn chương trình S (đoạn chương trình S thoả đặc tả).
+ Hoặc để đặc tả ngữ nghĩa đoạn chương trình (thực hiện sưu liệu chương trình) nhằm mục đích làm người đọc hiểu được ý nghĩa của đoạn chương trình

III. NGÔN NGỮ LẬP TRÌNH.

Để kiểm chứng tính đúng của một đoạn chương trình, đầu tiên cần trình bày đoạn chương trình đó trong một dạng ngôn ngữ lập trình chuẩn mực ở dạng cốt lõi.

Ngôn ngữ lập trình ở dạng cốt lõi chỉ bao gồm các thao tác chuẩn : lệnh gán, lệnh điều kiện, lệnh lặp while và lệnh ghép (dãy tuần tự các lệnh).

Cú pháp của ngôn ngữ cốt lõi được định nghĩa trong dạng BNF như sau :

```
< lệnh >          ::= < lệnh đơn > | dãy lệnh
< lệnh đơn >      ::= < lệnh gán > | < lệnh điều kiện > | < lệnh lặp >
< dãy lệnh >      ::= < lệnh đơn > | < lệnh đơn > ';' < dãy lệnh >
< nhóm lệnh >     ::= < lệnh đơn > | 'begin' < dãy lệnh > 'end'
< lệnh gán >      ::= <biến> ':=' < biểu thức >
< lệnh điều kiện > ::= 'if' < biểu thức > 'then' < nhóm lệnh > 'else' < nhóm lệnh > |
                       'if' < biểu thức > 'then' < nhóm lệnh >
< lệnh lặp >      ::= 'while' < biểu thức > 'do' < nhóm lệnh >
```

Định nghĩa trên xác định rằng mỗi < lệnh > mà ta khảo sát có thể là :
- <Lệnh đơn> : bao gồm các trường hợp :
  + < Lệnh gán > Ví dụ : Y := (X + Y) * Z ;
  + < Lệnh điều kiện > mà < nhóm lệnh> sau 'then' hay 'else' có thể là một <lệnh đơn> hay một <dãy lệnh> được bắt đầu bởi 'begin' và chấm dứt bởi 'end'.

Ví dụ :
```
if (x > 0) then y := z
           else begin z := x * 2 ;
                      if( z = y) then y := 0
                end ;
```
  + < Lệnh lặp > với một < biểu thức > biểu thị điều kiện lặp và < nhóm lệnh>

```
Ví dụ  :   while (x > 0) do    begin   y  :=  x ;
                                        while ( y > 0)  do  y  :=  y -1 ;
                                         x  :=  x - 1 ;
                                    end ;
```

- <Dãy lệnh> chính là dãy tuần tự các <lệnh đơn> ngăn cách bởi dấu ';'

CHƯƠNG V
KIỂM CHỨNG TÍNH ĐÚNG CÓ ĐIỀU KIỆN

I. CÁC KHÁI NIỆM VỀ TÍNH ĐÚNG.

Xét bộ 3 gồm : đọan lệnh S, các tân từ trên các biến của chương trình (có thể bao gồm các biến giả) P, Q (P mô tả điều kiện đầu , Q mô tả điệu kiện cuối).

Bộ 3 : { P } S { Q } tạo nên đặc tả đoạn lệnh S .

Đặc tả : { P } S { Q } được gọi là thỏa đầy đủ (đúng đầy đủ – đ đ đ) nếu xuất phát từ bất kỳ 1 trạng thái thỏa P thực hiện đoạn lệnh S thì việc xử lý sẻ dừng ở trạng thái thỏa Q .

Đặc tả : { P } S { Q } được gọi là thỏa có điều kiện (đúng có điều kiện – đcđk) nếu xuất phát từ bất kỳ 1 trạng thái thỏa P thực hiện đoạn lệnh S nếu việc xử lý dừng thì trạng thái cuối thỏa Q (tính dừng của S chưa được khẳng định).

Khẳng định { P } S { Q } diễn đạt tính đúng có điều kiện (condition correctness) (tđcđk) của S. Tính đúng của S dựa trên đkđ P và đkc Q với giả định rằng tính dừng của S đã có.

Ví dụ : a) { $(x = x_o)$ and $(y = y_o)$ } — Nếu $(x = x_o)$ và $(y = y_o)$ trước khi
t := x — t := x được thi hành
{$(t = x = x_o)$ and $(y = y_o)$ } — Thì sau đó $(t = x = x_o)$ và $(y = y_o)$

b) {$(t = x = x_o)$ and $(y = y_o)$ } — Nếu $(t = x = x_o)$ và $(y = y_o)$ trước khi
x := y — x := y được thi hành
{ $(t = x_o)$ and $(x = y = y_o)$ } — Thì sau đó $(t = x_o)$ và $(x = y = y_o)$

c) { $(t = x_o)$ and (x = y = yo) } — Nếu $(t = x_o)$ và $(x = y = y_o)$ trước khi
y := t — y := t được thi hành
{$(y = x_o)$ and $(x = y_o)$ } — Thì sau đó $(y = x_o)$ và $(x = y_o)$

Các phát biểu a, b, c là đúng theo cảm nhận của ta về lệnh gán.

d) { $x > 0$ } — Nếu $(x > x_o)$ trước khi
x := x-1 — x := x-1 được thực hiện
{ $x > 0$ } — Thì sau đó $(x > 0)$

Phát biểu d là sai vì có một trạng thái ban đầu x = 1 thoả P ($x > 0$) nhưng sau khi thi hành lệnh x := x -1 (x giảm 1) thì x = 0 không thoả Q ($x > 0$) .

II. HỆ LUẬT HOARE (HOARES INFERENCE RULES).

Để có thể thực hiện chứng minh hình thức về tính đúng của các đoạn chương trình, ta cần có những tiền đề mô tả tác động của các thao tác xử lý cơ bản (lệnh cơ bản) của ngôn ngữ dùng viết chương trình (ở đây là ngôn ngữ cốt lõi đã được giới thiệu ở IV.3). Một hệ tiên đề có tác dụng như thế của Ca. Hoare , được trình bày dưới dạng một hệ luật suy diễn (inference rules) được xét dưới đây .

1. Các luật hệ quả (Consequence rules)

1a.

$$\frac{P \Rightarrow Q ,\ \{ Q \}\ \ S\ \ \{ R \}}{\{ P \}\ \ S\ \ \{ R \}} \qquad (1a)$$

Nếu đkđ P mạnh hơn điều kiện Q .Tức là: P ⇒ Q hay { P } ⊆ { Q } (tập hợp các trạng thái thoả P là tập con của các tập trạng thái thoả Q) và mỗi trạng thái thoả Q đều đảm bảo trạng thái sau khi thi hành S (với giả định S dừng) thoả R thì mỗi trạng thái thoả P đều đảm bảo trạng thái sau khi thi hành S (với giả định S dừng) thoả R.

Ví dụ 1 : Kiểm chứng tđcđk đặc tả sau :

{ x = 3 } x := 5 ; y := 2 { x = 5, y = 2 }

Ta có : { true} x := 5 ; y := 2 { x = 5 ; y = 2 } (a) // tạm công nhận

và (x = 3) => true (b) // hiển nhiên

Nên { x = 3 } x := 5 ; y := 2 { x = 5, y = 2 } // theo tiên đề (1a)

Ví dụ 2 : Kiểm chứng tđcđk đặc tả sau :

{ x > 3 } x := x -1 { x > 0 }

Ta có : { x > 1 } x := x-1 { x > 0 } (a) //tạm công nhận

và (x > 3) => (x > 1) (b) // hiển nhiên

Nên { x > 3 } x := x -1 { x > 0 } // theo tiên đề (1a)

1b.

$$\frac{Q \Rightarrow R ,\ \{ P \}\ \ S\ \ \{ Q \}}{\{ P \}\ \ S\ \ \{ R \}} \qquad (1b)$$

Ví dụ 3 : Kiểm chứng tđcđk đặc tả sau :

{ true } x := 5 ; y := 2 { odd(x) and even(y) }

Ta có : { true } x := 5 ; y := 2 { (x = 5) , (y = 2) } (a) // tạm công nhận

và ((x = 5) and (y = 2)) => odd(x) and even(y) (b) // hiển nhiên

Nên { true } x := 5 ; y := 2 { odd(x) and even(y) } //theo (1b)

Ví dụ 4 : Kiểm chứng tđcđk đặc tả :

{ x > 1 } x := x -1 { x >= 1 }

Ta có : { x > 1 } x := x-1 { x > 0 } (a) // tạm công nhận

và (x > 0) => (x >= 1) // (b) // vì x là biến nguyên

Nên { x > 1 } x := x -1 { x >= 1 } // theo (1b)

Hai luật này cho phép liên kết các tính chất phát sinh từ cấu trúc chương trình với các suy diễn logic trên dữ kiện.

2. Tiên đề gán (The Assignement Axiom)

{ P(bt) } x := bt { P(x) }	(2)

Từ (2) ta suy ra nếu trước lệnh gán x := bt ; trạng thái chương trình làm P(bt) sai (thoả not P(bt)) thì sau lệnh gán P(x) cũng sai (thỏa notP(x)).

Lệnh gán x := bt xoá giá trị cũ của x , sau lệnh gán x mang giá trị mới là trị của biểu thức bt , còn tất cả các biến khác vẫn giữ giá trị như cũ.

Ví dụ : Tính đúng có điều kiện của các đặc tả sau được khẳng định dựa vào tiên đề gán

a) { x = x } y := x { x = y }

b) { 0 <= s + t - 1 } s := s + t - 1 { 0 <= s }

c) { i = 10 } j := 25 { i = 10 }

3. Các luật về các cấu trúc điều khiển .

a) Luật về dãy lệnh tuần tự (Rules on Sequential Composition)

$$\frac{\{ P \}\ S_1\ \{ R \},\ \{ R \}\ S_2\ \{Q \}}{\{ P \}\ S_1\ ;\ S_2\ \{ Q \}} \qquad (3.1)$$

Giả định có tính dừng của S_1 và S_2, luật này phát biểu ý sau :

Nếu: i) Thi hành S_1 với đkđ P đảm bảo đkc R (đặc tả { P } S_1 { R } đ cđ k)

ii) Thi hành S_2 với đkđ R đảm bảo đkc Q (đặc tả { R } S_2 { Q } đ cđ k)

Thì : thi hành S ≡ S_1 ; S_2 với đkđ P đảm bảo đkc Q (đ ặc tả { P } S_1 ; S_2 { Q } đ cđ k)

Ví dụ : Kiểm chứng tđcđk đặc tả :

{true} x := 5 ; y := 2 { x = 5, y = 2 }
Ta có : { 5 = 5 } x := 5 { x= 5} (a) // tiên đề gán
true =>(5 = 5) và (x = 5) => ((x = 5) and (2 = 2)) (b) // hiển nhiên
{true} x := 5 {(x = 5) and (2 = 2) } (c) //theo luật hệ qủa
{ x = 5 , 2 = 2 } y := 2 {(x = 5) and (y = 2) } (d) // tiền đề gán
{true} x := 5 ; y := 2 { x = 5, y = 2 } // theo luật tuần tự

b) Luật về điều kiện (chọn) (Rule for conditionals)

b1)

$$\frac{\{ P \text{ and } B\}\ S_1\ \{Q\},\ \{ P \text{ and } (\text{not } B)\}\ S_2\ \{ Q \}}{\{ P \}\ \text{if } B \text{ then } S_1 \text{ else } S_2\ \{ Q \}} \qquad (3.2a)$$

Ý nghĩa luật này là :

Nếu có :

{ P and B } + Nếu xuất phát từ trạng thái thỏa P and B
S_1 thi hành S_1 thì sẻ tới trạng thái thỏa Q
{ Q }

Và

{ P and notB } + Nếu xuất phát từ trạng thái thỏa P and not B
S_2 thi hành S_2 thì sẻ tới trạng thái thỏa Q
{ Q }

Thì suy ra :

{ P } Nếu xuất phát từ trạng thái thỏa P
if B then S_1 else S_2 thi hành lệnh if B then S_1 else S_2
{ Q } thì sẽ tới trạng thái thỏa Q .

b2)

$$\frac{\{ P \text{ and } B\}\ S\ \{ Q\}\ ,\ P \text{ and } (\text{not } B) \Rightarrow Q}{\{ P\}\ \text{if } B \text{ then } S\ \{ Q\}} \qquad (3.2b)$$

Ví dụ1 : Kiểm chứng tđcđk đặc tả :

{ i> 0 } if (i= 0) then j := 0 else j := 1 {j=1}

Ta có : ((i> 0) and (i = 0)) ≡ false (a) //hiển nhiên
{ (i> 0) and (i = 0)} j := 0 {j=1} (b) //{false} S { Q } đúng với ∀ S , Q
((i> 0) and not(i = 0)) ≡ true (c) // hiển nhiên

{true } j := 1 { j=1 } (d) //tiên đề gán

{(i >0) and not(i = 0)} j := 1 {j=1} (e) // c,d ,luật hệ qủa

Từ b ,e và tiên đề 3.2a ta suy ra ĐPCM.

Ví dụ2 : Kiểm chứng tđcđk đặc tả :

{i >= j - 1} if (i > j) then j := j+1 else i := i+1 {i >= j}

Ta có : {i >= j+1} j := j+1 {i >= j} (a) //tiên đề gán

((i >= j-1) and (i > j)) ==> (i >= j+1) (b) // biến đổi với chú ý i , j nguyên

{(i >= j-1) and (i > j)} j := j + 1 {i >= j} (c) // a,b ,luật hệ quả

{i+1 >= j} i := i+1 {i >= j} (d) // tiên đề gán

((i >= j-1) and not(i > j)) ==> (i+1 >= j) (e) // biến đổi

{(i >= j-1) and not(i > j)} i := i + 1 {i >= j} (g) // d ,e , luật hệ quả)

Từ c , g dựa vào 3.2a suy ra ĐPCM.

Ví dụ 3 : Kiểm chứng tđcđk đặc tả :

{true} if odd(x) then x := x+1 {even(x)}

Ta có : {even(x+1)} x := x+1 {even(x)} (a) //tiên đề gán

true and odd(x) ==> even(x+1) (b) // hiển nhiên

{true and odd(x)} x := x+1 {even(x)} (c) // a ,b , luật hệ quả

true and not odd(x) ==> even(x) (d) // hiển nhiên

Từ (c) và (d) dựa vào 3.2b suy ra ĐPCM .

b3) Luật về lệnh lặp While

$$\frac{\{\ I \text{ and } B\ \}\quad S \quad \{\ I\ \}}{\{\ I\ \}\ \text{ while } B \text{ do } S\ \ \{\ I \text{ and } (\text{not } B\ )\ \}} \qquad (3.3)$$

Luật này nói rằng nếu I không bị thay đổi bởi một lần thực hiện lệnh S thì nó cũng không bị thay đổi bởi toàn bộ lệnh lặp While B do S. Với ý nghĩa này I được gọi là bất biến (invariant) của vòng lặp.

Chú ý rằng khẳng định : { P } while B do S { Q } thỏa dựa vào hệ luật Hoare chỉ xác định tđcđk (conditionnal correctness). Để chứng minh tính đúng (correctness) thực sự ta cần bổ sung chứng minh lệnh lặp dừng.

Ví dụ1 :

Kiểm chứng tính đúng có điều kiện của đặc tả :

{i<=n} while (i < n) do i := i+1 {i=n}

Xem I là (i <= n) thì :

{ I and (i < n)} i := i+1 {I} (a) // dễ kiểm chứng

Nên {I} while (i < n) do i := i+1 { I and not(i<n)} (b) // luật 3.3

Mà I and not(i<n) ≡ (i <= n) and (i >= n) ==> i=n (c)

Từ b ,c , luật hệ qủa ta có ĐPCM.

Ví dụ 2 : Kiểm chứng tính đúng có điều kiện của đặc tả :

{sum = 0 , i = 0 , n > 0}

while (i <> n) do begin

i := i+1 ; sum := sum+i // S

end ;

{sum = n * (n+1)/2} // tức sum = 1 + 2 + ... + n

Ở đây : I là (sum = i*(i+1)/2) ; B ≡ (i <> n)

Ta có :

{(sum = i*(i+1)/2) ,(i<>n)} i := i+1 ; sum := sum+i {sum = i*(i+1)/2}

(a) //tiên đề gán và tuần tự

{ I } while B do S { I and not B } (b) // a,và luật 3.3

(s = 0) and (i = 0) and (n >0) ==> s = i*(i+1)/2 (c) //hiển nhiên

(s = i*(i+1)/2) and not(i<>n)) ==> s=n*(n+1)/2 (d) //hiển nhiên

Từ b , c , d ta suy ra ĐPCM.

III. KIỂM CHỨNG ĐOẠN CHƯƠNG TRÌNH KHÔNG CÓ VÒNG LẶP.

Cho : P, Q là các tân từ trên các biến của chương trình , S là một lệnh tổ hợp từ các lệnh gán với cấu trúc điều kiện và tuần tự. Chứng minh đặc tả : { P } S { Q} đúng đầy đủ .

Ở đây vì mỗi lệnh chỉ được thi hành một lần nên tính dừng của đoạn lệnh S được suy ra từ tính dừng của lệnh gán mà luôn được xem là hiển nhiên . Vì vậy trong trường hợp này tính đúng có điều kiện trùng với tính đúng đầu đủ.

1) Bài toán 1 : S là dãy tuần tự các lệnh gán .

Ví dụ1 : Kiểm chứng tính đúng của đoạn lệnh hoán đổi nội dung 2 biến x và y

a) {($x=x_o$) and ($y = y_o$) }

t := x ; x := y ; y := t ;

{($x=y_o$) and ($y = x_o$) }

Chứng minh

{($x = y_o$) and ($t = x_o$) } y := t {($x = y_o$) and ($y = x_o$) } (a) // tiên đề gán

{($y = y_o$) and ($t = x_o$) } x := y {($x = y_o$) and ($t = x_o$) } (b) // tiên đề gán

{($y = y_o$) and ($t = x_o$) } x := y ; y := t {($x = y_o$) and ($y = x_o$) } (c) // a , b , luật tuần tự

{($y = y_o$) and ($x = x_o$) } t := x {($y = y_o$) and ($t = x_o$) } (d) // tiên đề gán

(($x = x_o$) and ($y = y_o$)) ≡ (($y = y_o$) and ($x = x_o$) } (e) // giao hoán

{($x = x_o$) and ($y = y_o$) } t := x {($y = y_o$) and ($t = x_o$) } (g) // d ,e, luật hệ quả

$\{(x = x_o) \text{ and } (y = y_o)\}$ t := x ; x := y ; y := t $\{(x = y_o) \text{ and } (y = x_o)\}$ (h) // c ,g , luật tuần tự

Ví dụ 1 : Kiểm chứng tính đúng của đặc tả :

{ (m :: k=2*m) and $(y * z^k = c)$}
k := k div 2 ;
z := z * z ;
{ $y * z^k = c$}

Chứng minh :

(a) {$y * (z*z)^k = c$} z := z * z {$y*z^k = c$} (tiên đề gán)

(b) {$y * (z*z)^{k \text{ div } 2} = c$} k := k div 2 {$y*(z*z)^k = c$} (tiên đề gán)

(c) {$y * (z*z)^{k \text{ div } 2} = c$} k := k div 2 ; z := z*z {$(y*z)^k = c$} (a ,b , luật tuần tự)

(d) (m :: k = 2*m) and ($y * z^k = c$) ==> ($y*z^{2m} = c$) and (m = k div 2)
==> $y * (z*z)^{k \text{ div } 2} = c$

c ,d , luật hệ quả suy ra ĐPCM.

Nhận xét :

Với dãy tuần tự các lệnh gán, việc chứng minh {P} S_1; ...;S_n {Q} thường được bắt đầu từ lệnh cuối cùng, dùng tiên đề gán để được đkđ, rồi cứ thế lần ngược về đến S1.

{P_n} S_n {Q} (n) tìm P_n từ S_n ,Q và tiên đề gán

{P_{n-1} } S_{n-1} {P_n } (n-1) tìm P_{n-1} từ S_{n-1} , P_n và tiên đề gán

{P_{n-1} } S_{n-1} ; S_n {Q} luật về dãy lệnh tuần tự

...

...

{P_1 } S_1 ;...; S_n {Q} (1) sau n-1 lần tương tự như trên.

Sau đó dùng các tính chất của dữ kiện chứng minh logic rằng :

P ==> P_1 (0)

Từ (1) , (0) ,dựa vào luật hệ quả ta có : {P} S_1 ; ... ; S_n {Q} (ĐPCM)

2) Bài toán 2 :

a) Kiểm chứng đặc tả : {P} if B then S_1 else S_2 {Q}

Với S_1, S_2 là nhóm các lệnh gán , B là biểu thức boolean.

Cách chứng minh :

+ Bước 1 : Tìm P_1, P_2 thỏa : {P_1} S1 {Q} (1a)
{P_2} S2 {Q} (1b)

+ Bước 2 : Chứng minh (dùng các tính chất logic và đại số)

P and B ==> P_1 (2a)

P and (not B) ==> P_2 (2b)

+ Bước 3 : Dùng luật hệ quả suy ra :

{P and B} S_1 {Q} (3a) // 1a ,2a , và luật hệ qủa

{P and (not B)} S_2 {Q} (3b) // 1b ,2b , và luật hệ qủa

+ Bước 4 : Dũng (3a) , (3b) , luật điều kiện suy ra :

{P} if B then S_1 else S_2 {Q} (ĐPCM)

b) Kiểm chứng đặc tả : {P} S_0 ; if B then S_1 else S_2 {Q} (*)

với S_1, S_2, S_0 là dãy các lệnh gán

Ví dụ : Kiểm chứng đặc tả :

```
{y > 0}
x := y-1 ;
if (y > 3) then x := x*x
           else y := y-1
{x >= y}
```

Để khẳng định được (*) ta cần chỉ ra 1 khẳng định R mà :

{P} S_0 {R}

và {R} if B then S_1 else S_2 {Q} rồi dùng luật hệ quả để có (*)

Làm thế nào để tìm được R ? Do S_1 và S_2 là nhóm lệnh gán tuần tự nên ta có thể tìm được (bằng tiên đề gán và luật về dãy lệnh tuần tự) U và V để :

{U} S_2 {Q} và {V} S_3 {Q} .

Dĩ nhiên ta muốn U và V là các điều kiện tổng quát nhất có thể (ở đây là yếu nhất). R được xây dựng thế nào từ U và V ? Khả năng tổng quát nhất cho R để sau khi điểm điều kiện B sẽ có được U hoặc V là : R ≡ (B ==> U) and (not B ==> V)

Như sau này sẻ chỉ ra U , V , R được xây dựng như vậy là yếu nhất (weakest precondition) để đạt được Q tương ứng với lần lượt các lệnh S_1 , S_2 và if B then S_1 else S_2 , và được ký hiệu là : WP(S_2,Q) ,WP(S_3,Q) và WP(if B then S_2 else S_3, Q) tương ứng.

Ví dụ 1 : Kiểm chứng đặc tả :

```
{ y > 0 }
 x := y - 1 ;
 if ( y > 3 ) then  x := x * x
                else  y := y - 1 ;
{ x >= y }
```

Trong ví dụ này :

P là tân từ : (y > 0) ; Q là tân từ : (x >= y)

B là biểu thức boolean : (y > 3)

S_0 là lệnh gán : x := y - 1 ;

Do S_1 và S_2 là lệnh gán : x := x * x ;

S_2 là lệnh gán : y := y - 1 ;

Ta có :

{x^2 >= y} x := x*x {x >= y} suy ra U ≡ WP(S_1 , Q) ≡ x^2 >= y (a)

{x >= y-1} y := y-1 {x >= y} suy ra V ≡ WP(S_2 , Q) ≡ x >= y-1 (b)

Đặt R ≡ (B ==> U) and (not B ==> V)

≡ ((y > 3) ==> (x^2 >= y)) and ((y <= 3) ==> (x >= y-1))

Ta chứng minh được dễ dàng.

R and (y>3) ==> (x^2 >= y) (c)

R and (not(y>3)) ==> (x >= y-1) (d)

nên theo luật hệ quả

{R and y>3} S_2 {x >= y} ({R and B} S_2 {Q}) (e)

{R and not(y>3)} S_3 {x >= y} ({R and (not B)} S_3 {Q}) (g)

Theo luật về lệnh chọn

{R} if (y>3) then x := x*x else y := y-1 {x >=y} (h)

theo tiên đề gán.

{ ((y>3) ==> ((y-1) 2 >=y)) and ((y <=3) ==> ((y-1) >= (y-1))) }

x := y -1

{ R } (i)

Dễ kiểm chứng được

(y>3) ==> ((y-1) 2 >= y ≡ true (j)

(y<=3) ==> (y-1) >= y-1 ≡ true (k)

nên

(y >0) ==> ((y>3) ==>($(y-1)^2$ >=y)) and ((y<=3) ==> ((y-1) >=(y-1))) (l)

Theo luật hệ quả

{y > 0}

x := y-1;

if (y>3) then x := x*x else y := y-1

{ x >= y} (m) // ĐPCM

Ví dụ 2 : kiểm chứng đặctả :

{ true }

if (i <= j) then if (j<k) then m := k else m := j

else if(i<k) then m := k else m := i

{(m >= i) and (m >= j) and (m >= k)}

Đặt Q(m) ≡ (m >= i) and (m >= j) and (m >= k)

Ta có :

(a) {Q(i)} m := i {Q(m)} (tiên đề gán)

(b) {Q(k)} m := k {Q(m)} (tiên đề gán)

Đặt R1 ≡ ((i < k) ==> Q(k)) and ((i >= k) ==>Q(i))

dùng luật về lệnh chọn ta sẽ chứng minh được :

{R1} if (i < k) then ... {Q(m)} (c)

Tương tự đặt R2 ≡ (j<k) ==> (Q(k) and (j >= k)) ==> Q(j)

Ta có: {R2} if (j < k) then ... {Q(m)} (d)

Dùng biến đổi đại số và logic để chứng minh

(true and (i <= j)) ==> R2 (e)

(true and (i > j)) ==> R1 (g)

Từ c , d, e ,g , theo luật về lệnh chọn ta có ĐPCM.

Nhận xét :

Để chứng minh đặc tả {P} S {Q} với S là tổ hợp lệnh gồm chỉ các lệnh gán và điều
kiện đúng đầy đủ ,ta thực hiện công việc xây dựng điều kiện đầu yếu nhất P_1 của S ứng với Q , sau đó bước kiểm chứng cuối cùng chỉ đơn giản là chứng minh : P ==> P_1. Công việc trên được trình bày dưới dạng một hàm đệ quy như sau :

```
function  DKDYN (S : nhóm_lệnh ; Q : tân_từ ) : tân_từ ;
   var  t : câu lệnh ;
    begin
      if  (S <> rỗng ) then  begin
                                t :=  lệnh_cuối(S);
                                S :=  S – t ;
                                if ( t = lệnh_gán(x:=bt)) then DKDYN  :=  DKDYN(S,Q(
x=bt) )
                                     else    (*  t là lệnh if  *)
                                 DKDYN := (điều_kiện(t)==>DKDYN(phần_đúng(t),Q))
                                and not (điều_kiện(t)==>DKDYN(phần_khong đúng(t),Q))
                             end
                        else    DKDYN := Q
         end ;
```

IV. KIỂM CHỨNG ĐOẠN CHƯƠNG TRÌNH CÓ VÒNG LẶP.

1. Bất biến

Một tính chất đặc thù của trí tuệ là nó thoát khỏi công việc mà nó đang thực hiện, khảo sát kết quả mà nó đã làm và luôn luôn tìm kiếm, và thường phát hiện được, các khuôn mẫu (Douglas R. Hofstadter).

Một bất biến là một tính chất không thay đổi tồn tại trong một khung cảnh, một sự kiện một quá trình thay đổi thường xuyên.

Một điều có vẻ nghịch lý là trong một thế giới, thay đổi và cần thiết phải thay đổi nhanh chóng, các bất biến lại có ý nghĩa rất quan trọng đối với chúng ta.

Một em bé trong một nước nói tiếng Anh học cách thành lập dạng số nhiều của danh từ : dogs, cats, hands, arms ..., cách thành lập dạng quá khứ của động từ :

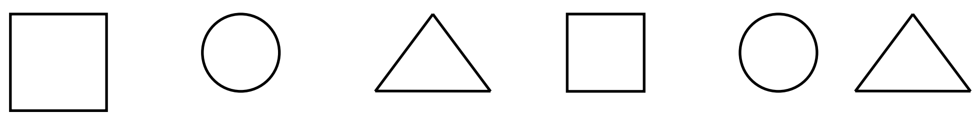
kicked, jumped, walked ... bằng học luật không đổi (thêm s, thêm ed), kèm theo với việc học thuộc một số trường hợp ngoại lệ. Hãy tượng tượng việc học sẽ khó như thế nào nếu không có các luật không đổi (bất biến) này.

Việc nhận thức được các bất biến thường dẫn tới những lời giải đơn giản cho các bài toán khó.

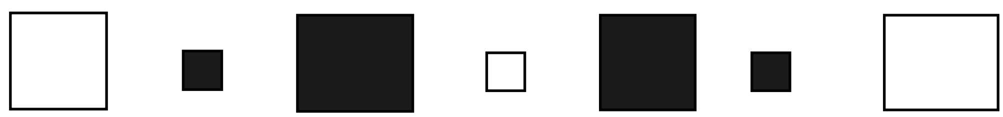
Đầu óc con người dường như có một khả năng đặc biệt để nhận thức các bất biến hay các "khuôn mẫu".

Hãy quan sát 2 dãy các hình sau :

(a)



(b)



Hình kế tiếp trong mỗi dãy hình trên là gì ? Tính chất bất biến của mỗi dãy là gì ?

(a) Lặp lại bộ 3 hình vuông, tròn, tam giác.

(b) Về dạng thì là sự lặp lại của cặp 2 hình vuông lớn và nhỏ. Về màu thì là sự lặp lại của một màu trắng và 2 màu sậm.

Trong lĩnh vực chương trình cho máy tính, ta cũng cần nhận thức các sự việc bằng cách phát hiện các bất biến. Đối với một chương trình, ta có nhiều lần máy tính thi hành nó, mỗi lần thi hành được gọi là một quá trình (process) và tác động trên các dữ kiện khác nhau. Tính bất biến của các quá trình này chính là đặc tả của chương trình.

Bên trong một chương trình có thể có các vòng lặp. Việc thực hiện vòng lặp làm biến thiên nhiều lần trạng thái các biến chương trình (các đối tương dữ liệu) , mà số lần biến thiên thường không biết trước được. Làm thế nào để hiểu được tác động của vòng lặp và đi đến chứng minh vòng lặp thực hiện một tính chất (giữ một bất biến) nào đó thể hiện bởi đặc tả của nó.

Mô hình biến đổi trạng thái chương trình của vòng lặp **while B do S**



{ P } là trạng thái trước vòng lặp .

{ P_i } là trạng thái sau lần lặp thứ i .

{ Q } là trạng thái sau vòng lặp .

Việc nhận thức (tìm ra) các tính chất bất biến của trạng thái chương trình trước và sau mỗi lần lặp có vai trò quyết định ở đây.

Ví dụ : với vòng lặp :

```
tg := 0 ;
 i := 0 ;
while ( i <= n ) do    begin
                          i := i + 1 ;
                         tg := tg + a[i] ;
                       end ;
```

Tính chất bất biến ở đây là : bất chấp i, sau lần lặp thứ i, tg sẽ chứa tổng i phần tử đầu tiên của array a(a[1], a[2], ..., a[i]).

Tức là : tg = S(j: 1 <= j <=i : a[j]) = $\sum_{1}^{i} a[j]$

2. Lý luận quy nạp và chứng minh bằng quy nạp.

Trong khoa học cũng như trong đời sống hàng ngày, người ta thường cần phải suy diễn từ các phát hiện riêng lẻ để đi đến các quy luật (bất biến) phổ dụng cho mọi(hay hầu hết) trường hợp có thể.

Quá trình mà con người xác lập được một tính chất bất biến từ một tập hợp các quan sát được gọi là suy diễn quy nạp.

Suy diễn quy nạp xuất phát từ quan sát và kết quả là cho ra các giả thuyết cần chứng minh.

Ví dụ 1 : từ các quan sát :

$1 = 1 = 1^2$

$1 + 3 = 4 = 2^2$

$1 + 3 + 5 = 9 = 3^2$

$1 + 3 + 5 + 7 = 16 = 4^2$

Bằng quy nạp người ta đặt giả thuyết : $1 + 3 + ... (2*n - 1) = n^2$

Ta có thể thử lại giả thuyết này với n = 5, 6.... Tuy nhiên, để khẳng định rằng giả thuyết đúng với mọi n, ta cần có chứng minh. Phương pháp chứng minh thường dùng trong trường hợp này là chứng minh bằng quy nạp.

a) Nguyên lý quy nạp toán học đơn giản .

Để chứng minh một tân từ P(n) phụ thuộc vào số tự nhiên là đúng với mọi n .

Ta cần chứng minh 2 điều sau :

(i) P(0) là đúng

(ii) Nếu P(n) được giả định là đúng thì sẽ suy ra P(n+1) cũng đúng.

Khẳng định P(0) được gọi là cơ sở (basis) và bước chứng minh (ii) là bước quy nạp (inductive step). Khi có được 2 điều (i) và (ii), dựa vào nguyên lý quy nạp toán học, ta kết luận rằng P(n) đúng với mọi số tự nhiên n .

Trên thực tế nguyên lý trên thường được áp dụng hơi khác.

+ Để chứng minh P(n) đúng với mọi số tự nhiên n >= m thì cơ sở của chứng minh quy nạp là P(m) chứ không phải P(0).

+ Để chứng minh P(n) đúng với mọi số tự nhiên n thoả m <= n <= p ta chứng minh :

(i) P(m) đúng

(ii) Nếu m <= n < p và P(n) đúng thì P(n+1) đúng.

Ví dụ : (i) Cơ sở : P(1) chính là $1 = 1^2$ đúng

(ii) Giả sử P(n) đúng, tức là $1 + 3 + ... (2*m - 1) = n^2$

thì ta sẽ có :

$$1 + 3 + ... + (2*(n+1)-1) = (1+3+....+(2*n -1)) + (2*(n+1)-1)$$
$$= n^2 + 2*(n+1) - 1$$
$$= (n+1)^2$$

Vậy P(n+1) đúng .Dựa vào (i) và (ii), ta kết luận P(n) đúng với mọi số tự nhiên n >= 1 theo nguyên lý quy nạp toán học.

b) Nguyên lý quy nạp mạnh (Strong induction principle)

Để chứng minh P(n) đúng với mọi số tự nhiên n ta cần chứng minh hai điều sau :

(i) P(0) đúng

(ii) Nếu giả định là P(0), P(1), ... P(n) đều đúng thì P(n+1) cũng đúng

Cũng như nguyên lý quy nạp đơn giản, người ta có thể dùng các biến dạng của nguyên lý quy nạp mạnh để chứng minh P(n) đúng với mọi số tự nhiên n >= m cho trước hay với mọi số tự nhiên n mà m < n <= p với m,p cho trước.

3. Kiểm chứng chương trình có vòng lặp while.

a) Dạng tổng quát của bài toán .

Cho W là một lệnh lặp while B do S và cặp đkđ P, đkc Q.

Ta cần phải chứng minh rằng : đặc tả { P } W { Q } được thỏa đầy đủ .

Để chứng minh W thoả đầy đủ đặc tả P,Q ta cần chỉ ra 2 điều :

(i) P bảo đảm W dừng,tức là xuất phát từ trạng thái bất kỳ thoả P,thì hành W thì W sẽ dừng sau một thời gian hữu hạn (sau khi thực hiện hưu hạn lần lệnh S ở thân vòng lặp W thì B sẻ có gía trị false).

(ii) {P} W {Q} - Đúng có điều kiện (xuất phát từ trạng thái thỏa P sau khi thi hành W nếu W dừng thì sẻ đạt tới trạng thái thỏa Q).

Để chứng minh (ii) ta có thể dùng hệ luật Hoare mà chủ yếu là phải phát hiện được bất biến I.

Để chứng minh (i) W dừng ta cần dựa trên các biến bị thay đổi trong vòng lặp thường dựa vào một hàm f của các biến chương trình nhận gía tri nguyên và chỉ ra rằng :

(α) Ở đầu mỗi lần lặp (B thoả) thì f > 0 .

Tức là : I and B ==> f > 0

(β) Mỗi lần thực hiện S sẽ làm giảm thực sự giá trị của f.

Nếu (α) và (β) thoả thì S không thể lặp vô tận được (vì sau hưu hạn lần thì B sẻ nhận gía trị false).

Ví dụ : Chỉ ra các vòng lặp sau đây dừng :

```
    {n >= 0}
   k  :=  n ;
   while (k <>0 ) do    begin  {k > 0}
                           k  :=  k-1 ;
                           r  :=  2*r + p[k];
                           if (r >= q)  then  r :=  r - q
                        end ;
```

vì bất biến {k > 0} luôn được giữ đúng ở đầu vòng lặp. Ở đây hàm f chính là bằng k. f giảm sau mỗi lần lặp (vì k := k - 1). Vậy vòng lặp dừng .

```
  {x >= 0 ; y >= 0}
    a  :=  x ;
    b  :=  y ;
    while (a <>b)  do
             {max(a,b) > 0}
              if (a>b )  then   a  :=  a - b
                       else   b  :=  b - a
```

Ở đây hàm f = max(a,b). Ta luôn có bất biến max(a,b) > 0 ở đầu vòng lặp, f giảm sau mỗi lần lặp.

b) Các ví dụ về chứng minh chương trình có vòng lặp .

Ví dụ 1 : Xét đặc tả đoạn chương trình tính tích 2 số nguyên A và B với B >= 0 bằng phép cộng :

```
    { B >=  0 }
    R := 0 ;
    X := B ;
    while (X <>0 )  do     begin
                              R :=  R + A ;
                              X :=  X - 1 ;
                           end ;
         { R   =  A*B  }
```

đkđ P ≡ B >= 0

đkc Q ≡ R = A * B

Bước 1: Kiểm chứng tính đúng có điều kiện của đặc tả {P} S {Q}

+ Kiểm chứng đoạn lệnh trước vòng lặp : Chứng minh đặc tả sau đúng .

```
     { B >=  0 }
       R := 0 ;          (*)
      X := B ;
     {X = B , R = 0,  B >=  0 }
```

Ta có :

{0 = 0 , B >= 0} R := 0 {R = 0 , B >= 0} (1) tiên đề gán

{0 = 0 , B >= 0} ⇒ { B >= 0} (2) hiển nhiên

{B >= 0} R := 0 {R = 0, B >= 0} (3) luật hệ quả dựa vào (1),(2)

{B = B , R = 0 , B >= 0} X := B {X = B , R = 0 , B >= 0} (4) tiên đề gán

{B =B , R =0 , B >= 0} ⇒ { R = 0 , B >= 0 } (5) hiển nhiên

{R = 0 , B >= 0} X := B {X := B , R = 0 , B >= 0} (6) Luật hệ quả dựa vào (4),(5)

{B >= 0} X := 0 ; X :=B {X = B , R = 0 , B >= 0} (7) luật tuần tự dựa vào (3),(6).

Như vậy với điều kiện đầu B >= 0 thì sau khi thực hiện xong 2 lệnh khởi động, ta có khẳng định X = B, R = 0, B >= 0 đặc tả (*) đúng .

+ Kiểm chưng vòng lặp :

- Phát hiện được bất biến của vòng lặp.

Bất biến ở đây là : " số lần X bị giảm đi chính là số lần A được cộng vào R "

Tức là : I ≡ (R = A*(B-X)) and (X >= 0)

Khẳng định (X >= 0) được thêm vào để chứng minh vòng lặp dừng.

- Chứng minh I là bất biến của vòng lặp :

{(R + A = A*B - A*X + A) and (X >0)}

R := R + A

{(R = A*B - A*X + A) and (X > 0)} (8) tiên đề gán

{ (R = A*(B - X)) and (X > 0)}

R := R + A

{ (R = A*B - A*X + A) and (X > 0)} (9) biến đổi từ (8)

{ (R = A*B - A*X + A) and (X > 0)} ≡

{ (R = A*(B -(X - 1))) and ((X – 1) >= 0)}(10)

{ (R = A*(B -(X - 1))) and ((X – 1) >= 0)}

X := X – 1

{(R = A*(B - X)) and (X>=0) } (11) tiên đề gán

{(R = A*B - A*X + A) and (X > 0)}

X := X – 1

{(R = A*(B – X)) and (X >= 0)} (12) luật hệ quả

{(R = A*(B –X)) and (X > 0)}

R : = R+A ; X := X - 1

{(R = A*(B – X)) and (X >= 0)} (13) luật tuần tự dựa vào (9),(12)

{(R = A*(B – X)) and (X >= 0) and (X<>0)} ==> {(R =A*(B – X)) and (X >0)}(14)

{(R = A*(B – X)) and (X >= 0) and (X<>0)}

R := R+A ; X := X-1

{(R = A*(B – X)) and (X >= 0)} (15) luật hệ quả dựa vào (13),(14)

Hay {I and C} R := R + A ; X := X – 1 {I}

với C là điều kiện vòng lặp X<>0

Do luật lặp ta có :

{I} while ... {I and not C}
(X=B) and (R=0) and (B>=0) ==> (R=A*(B-X)) and (X>=0) (16)
Kết hợp 15,16 và 5 dùng luật hệ quả rồi luật tuần tự, ta có :
{X=B and R=0 and B>=0} while ... {I and notC} (17)
{B>=0} R:=0 ; X:=B ; while ... {I and notC} (18)
Ở đây I and not C ≡ (R = A*(B-X)) and (X >= 0) and (X = 0)
mà (R = A*(B-X)) and (X >= 0) and (X=0) ==> R=A*B
Dùng luật hệ quả ta có đpcm
Bước 2 : Chứng minh tính dừng :
Đặt f = X, ta có :
(i) I and C ≡ (R = A*(B-X)) and (X >= 0) and (X<>0) => X > 0 => f > 0
(ii) Mỗi lần lặp, f bị giảm một đơn vị. Vậy vòng lặp phải dừng.
Từ (i) và (ii) ta kết luận được tính dừng từ bước 1 và bước 2 suy ra tính đúng đầy đủ của đoạn chương trình đối với P,Q.
Nhận xét từ chứng minh trên :
+ Đối với dãy các lệnh gán, nên phát xuất quá trình suy diễn từ điều kiện cuối.
+ Đối với vòng lặp cần xác định đúng bất biến của nó.
Chú ý : Ta có thể kiểm chứng tđcđk của đoạn chương trình trên bằng cách:
- Xây dựng một lược đồ chứng minh hợp lý bằng cách dựa vào các tiên đề và cảc khẳng định đã có trước đó chèn bổ sung các khẳng định trung gian ở những điểm khác nhau trong đoạn chương trình .

```
        {P } ≡ { B >= 0 }                (0)
    {(0 = A*(B – B )) and (B >= 0 )}          (3)
          R := 0 ;
     {(R = A*(B – B )) and (B >= 0)}            (2)
         X := B ;
    {I } ≡ {( R = A*(B – B )) and ( X >= 0 )} (1a)
        while (X <>0 ) do    begin
                   {I and C } ≡ {( R = A*(B – X )) and (X >= 0) and (X<>0)} (1b)
                   {(R + A = A*(B -(X – 1 )) and ( (X –1 ) >= 0)}  (5)
                              R := R+A ;
              {( R = A*(B -(X – 1 )) and ( (X - 1 ) >= 0)}      (4)
                              X := X - 1 ;
              {I } ≡ {(R = A*(B - X)) and ( X >= 0 )} (1d)
           end
      {I and notC } ≡ {( R = A*(B - X ) ) and (X >= 0) and not(X <> 0)} (1c)
                     {Q } ≡ { R = A*B} (6)
```

Lý lẽ để bổ sung là :
(1a) do phát hiện được bất biến I
(1b),(1c),(1d) dựa vào tiên đề về lệnh lặp xuất phát từ I
2, 3 dựa vào tiên đề gán xuất phát từ (1a)

4, 5 dựa vào tiên đề gán xuất phát từ (1d)

Các cặp khẳng định đi liền nhau là các điều kiện cần kiểm chứng :

(0) ==> (3) : (B >= 0) ==>((0 = A * (B-B)) and (B >= 0))

(1b) ==> (5) : ((R = A*(B – X)) and (X >= 0) and (X<>0)) ==> ((R + A = A*(B -(X – 1)) and ((X –1) >= 0))

(1c) ==> (6) : ((R = A*(B-X)) and (X >= 0) and (X = 0)) ==> (R=A*B)

Dễ dàng chứng minh các điều trên.

CHƯƠNG VI
KIỂM CHỨNG TÍNH ĐÚNG ĐẦY ĐỦ

I. CÁC KHÁI NIỆM.

1. Đặt vấn đề.

Ta thường gặp bài toán sau : Với tân từ Q trên các biến chương trình mô tả trạng thái cuối cần thỏa sau khi thực hiện lệnh S, tìm tập điêu kiện đầu thỏa đặc tả . Tức là với tân từ Q và đoạn lệnh S cho trước tìm tân từ P thỏa đầy đủ đặc tả : {P} S {Q}.

Dễ thấy rằng bài tóan sẻ có nhiều lời giải. Xuất phát từ một cặp gồm tân từ Q và đoạn lệnh S , có nhiều tân từ P thỏa .

Ví dụ :

Với Q ≡ (x > 0) ; S ≡ x := x - 1 ;

Các tân từ P sau đây đều thỏa :

(x > 1) , (x >= 5), (x > 5), ... , false

Mỗi tân từ P xác định một tập hợp các trạng thái. Trên tập hợp các trạng thái ứng cử này dĩ nhiên ta sẽ mong muốn chọn tập hợp lớn nhất có thể. Tức là ta quan tâm đến tân từ P là hạn chế yếu nhất trên không gian trạng thái . Dễ dàng thấy rằng ý nghĩa của quan hệ yếu ở đây là :

P yếu hơn Q tức là (Q ==> P)

hoặc { Q } ⊆ { P }

2. Định nghĩa WP(S,Q).

Nếu Q là một tân từ trên các biến chương trình và S là một đoạn lệnh thì điều kiện đầu yếu nhất của S dựa trên Q (the weakest precondition of S with respect to Q) là một tân từ trên các biến chương trình mô tả tập hợp mọi trạng thái ban đầu sao cho việc thi hành S bắt đầu ở một trạng thái thỏa nó đều được bảo đảm là sẽ dừng trong một trạng thái thoả tân từ cuối Q (thuộc tập {Q}),và được ký hiệu là WP(S ,Q)

Khái niệm WP là cơ sở cho việc mô tả một hệ thống quy tắc kiểm chứng tính đúng đầy đủ đoạn chương trình của Dijkstra . Ta sẽ tìm hiểu nôi dung của hệ thống này trong mối tương quan với hệ luật của Hoare.

Việc kết hợp các quy tắc của 2 hệ thống này sẽ cho ta một phương tiện hợp lý để chứng minh tính đúng đầy đủ của đoạn chương trình.

3. Hệ quả của định nghĩa.

+ Đặc tả { WP(S,Q) } S { Q } thỏa có điều kiện (đcđk)

+ WP(S,Q) bảo đảm tính dừng của S .Tức là S hoạt động đúng thực sự với đkđ WP(S,Q) và đkc Q .

+ WP(S ,Q) là tân từ yếu nhất thỏa đầy đủ đặc tả {P} S {Q} . Tức là nếu có tân từ P* bảo đảm S dừng và đặc tả {P*} S {Q} đúng có điều kiện thì P* ==> WP(S,Q) hay { P *} $\subseteq$ { WP(S,Q) } ({WP(S,Q) } là tập điều kiện đầu lớn nhất mà xuất phát từ đó thi hành S thì sẽ dừng tại trạng thái thỏa Q) .

Đây là các dấu hiệu đặc trưng để nhận ra WP(S,Q)

4. Các ví dụ.

Ví dụ 1 : Tính WP(while n<>0 do n := n – 1, n = 0) và so sánh với tân từ yếu nhất thỏa có điều kiện lệnh lặp while n<>0 do n := n – 1 với điều kiện cuối n = 0

+ Dựa vào quy luật của Hoare thì ta có :

{true} while n<>0 do n := n -1 { n = 0}

Thực vậy :

Từ : {true and (n<>0) } n := n-1 {true}

(xem bất biến vòng lặp là : I ≡ true)

ta suy ra : {true} while (n<>0) do n := n -1 {true and n = 0}

+ Từ đinh nghĩa WP ta suy ra :

wp (while (n<>0) do n := n -1 , n = 0) ≡ (n >= 0)

Ta có :

wp (while (n<>0) do n := n -1 , n = 0) ===> true

Tức là : tân từ yếu nhất thỏa đầy đủ đặc tả {P} S {Q} mạnh hơn tân từ yếu nhất thỏa có điều kiện đặc tả (tức là tập điều kiện đầu lớn nhất thỏa đầy đủ là tập con của tập điều kiện đầu thỏa có điều kiện)

Ví dụ 2 :

S ≡ i := 0 ; Q ≡ (i = 0) ;

Tìm wp (S,Q) .

Vì : + P ==> true với mọi P nên ta cũng có wp(S,Q) ==>true (a)

+ true bảo đảm S dừng và mọi trạng thái đầu đều dẫn đến Q nên

true ==> wp(S,Q) (b)

Từ (a),(b) ta suy ra : wp(i:=0, i = 0) ≡ true

Ví dụ 3 :

S ≡ i := 0 ; Q ≡ (i = 1) ;

Tính wp (S,Q) .

Đây là trường hợp ngược với ví dụ 2. Bất chấp trạng thái trước lệnh gán là gì, lệnh gán i := 0 không thể nào bảo đảm i=1.

Vì vậy : wp(i:= 0 , i=1) ≡ false

false mô tả tập hợp trạng thái rỗng. Tức là tập điều kiện đầu thỏa S,Q là tập rỗng .

II. TÍNH CHẤT CỦA WP.

Quan hệ giữa WP đối với các toán tử logic cấu tạo nên tân từ Q như thế nào?

1. Các quy ước :

a) Luật loại trừ trường hợp kỳ dị (The law of the excluded miracle).

WP(S,false) ≡ false

b) WP(S,true) là tân từ xác định tập các trạng thái bảo đảm tính dừng của S

Ví dụ : WP(while (n<>0) do n := n -1, true) ≡ (n >= 0)

2. Tính phân phối của and : wp(S,Q) and wp(S,R) ≡ wp(S,Q and R)

3. Tính phân phối của or : wp(S,Q or R) ≡ wp(S,Q) and wp(S,R)

4. Nếu Q ==> R thì wp(S,R) ==> wp(S,R)

III. CÁC PHÉP BIẾN ĐỔI TÂN TỪ.

1. Toán tử gán (tiên đề gán).

WP(x := bt , Q(x)) = Q(bt)

Ví dụ :

WP(i := i -1, i = 0) ≡ (i-1 = 0) ≡ (i = 1) .
WP(i := (l+u) div 2, l <= i <= u) ≡ l <= ((l+u) div 2) <= u
WP(i := 1, i =1) ≡ 1 = 1 ≡ true

2. Toán tử tuần tự.

WP(S1 ; S2 , Q) ≡ WP(S1 , WP(S2,Q))

Ví dụ :

WP(x := x+1 ; y := y+1 , x = y) ≡ WP(x := x+1 ; WP(y := y+1,x = y))
≡ WP(x:=x+1, x=y+1)
≡ x+1 = y+1 ≡ (x = y)

Quy luật này hàm ý rằng tổ hợp tuần tự các lệnh có tính kết hợp (associativity) tức là (S1 ; S2); S3 thì cũng cùng ý nghĩa với S1; (S2;S3).

Bởi vì với Q tuỳ ý wp((S1;S2);S3,Q) ≡ wp(S1 ; S2 , wp(S3,Q))
≡ wp(S1 , wp(S2, wp(S3,Q))) ≡ wp(S1 , wp(S2;S3,Q))
≡ wp((S1 ; (S2;S3)) ,Q)

Ví dụ :

Chứng minh tính đúng đầy đủ đặc tả sau :

{ S=i*(i+1)/2 }
i := i+1;
S := S+i;
{ S = i*(i+1)/2 }

Ta có :

wp(i := i+1 ; S := S+i, S=i*(i+1)/2)
≡ wp(i := i+1, wp(S := S+i, S=i*(i+1)/2))
≡ wp(i := i+1, S+i = i*(i+1)/2)
≡ S +i+1 = (i+1)*(i+1)+1)/2
≡ S = i*(i+1)/2)

Theo định nghĩa của wp ta có :

{ wp(i := i+1 ; S := S+i, S=i*(i+1)/2) }
i := i+1;
S := S+i;
{ S = i*(i+1)/2 }

đúng đầy đủ . Suy ra ĐPCM.

3. Toán tử điều kiện.

a)

WP(if B then S_1 else S_2, Q) ≡ (B ==> WP(S_1 , Q) and (not B ==> WP(S_2 , Q))

Ví dụ 1 :

Tính WP(if (i= 0) then j := 0 else j := 1, j=1)

Ta có : WP(j := 0, j = 1) ≡ (1 = 0) ≡ false
WP(j := 1 , j = 1) ≡ (1 = 1) ≡ true

Nên : WP(if (i = 0) then j := 0 else j := 1, j=1)
≡ ((i = 0) ==> false) and ((i<>0) ==> true)
≡ (not(i=0) or false) and true ≡ (i <> 0)

Ví dụ 2:

Tính WP(if (i>j) then j := j+1 else i := i+1, i >= j)

Ta có : WP(j := j+1, i >= j) ≡ i >= j+1 ≡ i > j
WP(i := i+1 , i >= j) ≡ i+1 >= j ≡ i >= j -1

Nên WP(if (i>j) then j := j+1 else i := i+1, i >= j)
≡ ((i > j) ==> (i > j)) and ((i <= j) ==> (i >= j -1))
≡ true and (not(i <= j) or (i >= j -1))
≡ (i > j) or (i >= j - 1) ≡(i > j)

b) Định lý sau đây chứng minh sự đúng đắn của toán tử điều kiện nếu chấp nhận hệ tiên đề của Hoare và tính dừng.

Định lý :

Gọi P ≡ (B==> WP(S_1,Q)) and (not B ==> WP(S_2, Q))

Thì ta có : + {P} if B then S_1 else S_2 {Q} đ cđk (1)

và + Với giả định S_1 và S_2 dừng nếu {R} if B then S_1 else S_2 {Q}

thì R ==> P (P là tân từ yếu nhất thỏa đầy đủ đặc tả) (2)

(Tức là : WP(if B then S_1 else S_2, Q) ≡ P

≡ (B==> WP(S_1,Q)) and (not B ==> WP(S_2 Q)) Chứng minh :

Theo định nghĩa của WP, nếu R ==> WP(S,Q) thì {R} S {Q} thỏa cđk

Mà ta có P and B ≡ B and WP(S_1,Q)) ==> WP(S_1,Q)

Vì vậy : {P and B} S_1 {Q} thỏa cđk

Tương tự {P and (notB)} S_2 {Q} thỏa cđk

Do đó, theo luật về lệnh chọn của Hoare, ta có : {P} if B then S_1 else S_2 {Q} (1)

Giả sử S_1 và S_2 luôn luôn dừng và {R} if B then S_1 else S_2 {Q}

Thì : {R and B} S_1 {Q}

{R and (not B)} S_2 {Q}

và : if B then S_1 else S_2 luôn luôn dừng.

Vì vậy theo định nghĩa của WP ta có : R and B ==> WP(S_1,Q)

và R and (notB) ==> WP(S_2,Q)

Hai khẳng định trên tương đương với : R ==> (B ==> WP(S_1,Q))

và R ==> (not B ==> WP(S_2,Q))

Vì vậy R ==> (B ==> WP(S_1,Q)) and (not B ==> WP(S_2,Q))) (2)

Từ (1) và (2) ta suy ra : P ≡ WP(if B then S_1 else S_2, Q) .

c) Ta cũng có khẳng định ngược lại : Nếu chấp nhận tiên đề :

WP(if B then S1 else S2, Q) ≡ (B ==> WP(S1, Q) and (not B ==> WP(S2,Q))

thì có thể chứng minh luật về lệnh chọn của Hoare là đúng :

Định lý : Giả sử S1, S2 dừng.

Nếu {P and B} S1 {Q}

và {P and not B1} S2 {Q}

thì {P} if B then S1 else S2 {Q} đúng

Chứng minh : (Bài tập)

4. Toán tử lặp.

a) Xây dựng WP(while B do S ,Q) .

Xét vòng lặp W ≡ while B do S , với đkc Q.

Xây dựng tân từ : WP(while B do S, Q)

Nó phải bảo đảm W dừng sau một số hữu hạn lần lặp lại S và tới trạng thái thỏa Q .

Gọi k là số lần lặp (số lần thực hiện lệnh S ở thân vòng lặp). Ta xây dựng quy nạp theo k :

Bước 0 : (k = 0) tân từ yếu nhất mô tả tập lớn nhất các trạng thái bảo đảm S lặp đúng 0 lần và tới trạng thái thỏa Q là : $P_o \equiv$ (not B) and Q

Bước 1 : (k = 1) tân từ yếu nhất mô tả tập lớn nhất các trạng thái bảo đảm S lặp đúng một lần và tới trạng thái thỏa Q là : $P_1 \equiv$ B and WP(S,P_o)
(tức là sau khi thực hiện một lần S thì trạng thái chương trình sẽ thoả P_o).

Bước 2 : (k = 2) tân từ yếu nhất mô tả tập lớn nhất các trạng thái bảo đảm S lặp đúng 2 lần và tới trạng thái thỏa Q là : $P_2 \equiv$ B and WP(S,P_1)
(tức là sau khi thực hiện một lần S thì trạng thái chương trình sẽ thoả P_1).

....

....

Bước k : Một cách tổng quát với k >= 1 thì tân từ yếu nhất mô tả tập lớn nhất các trạng thái bảo đảm S lặp đúng k lần và tới trạng thái thỏa Q là : $P_k \equiv$ B and WP(S,P_{k-1})

Như vậy một trạng thái đầu làm W dừng ở một trạng thái thoả Q khi và chỉ khi nó thoả khẳng định sau : $\exists(k : k >= 0 : P_k)$

Tức là :

$$\text{WP(while B do S , Q)} \equiv \exists(k : k >= 0 : P_k)$$

$$\text{Với } P_k = \begin{cases} \text{not B and Q} & \text{với } k = 0 \\ WP(S,P_{k-1}) & \text{với } k > 0 \end{cases}$$

Ví dụ :

Cho S là đoạn chương trình :

j := j* i ; k := k+j ; n := n+1 ;

và W là while (n <> m) do S

Q là : (k = (i^m+1 - 1) /(i-1) and j = i^m

(đoạn chương trình nhằm tính $k = 1 + i^1 + i^2 + ... + i^m$)

Giả sử rằng (i <> 0) và(i <> 1) , xác định WP(W,Q) .

Dãy các khẳng định P_n được xác định :

$P_o \equiv$ not(n <> m) and Q

$P_r \equiv$ (n <> m) and wp(S,P_{r-1}) với r = 1,2,3,...

Thực hiện tính toán ta có :

$P_o \equiv (n = m) \text{ and } (k = (i^{m+1} - 1)/(i -1)) \text{ and } (j = i^m)$

$P_1 \equiv (n <> m) \text{ and } (n+1 = m) \text{ and } ((k + j* i) = (i^m+1 - 1)/(i-1)) \text{ and } (j * i = i^m)$

$\equiv (n = m -1) \text{ and } (k = (i^m - 1)/(i-1)) \text{ and } (j = i^{m-1})$

Tương tự :

$P_2 \equiv (n = m - 2)$ and $(k = (i^{m-1} - 1)/(i-1))$ and $(j = i^{m-2})$

Ta có thể chứng minh bằng quy nạp giả thuyết sau (với mọi số tự nhiên r)

$P_r \equiv (n = m - r)$ and $(k = (i^{m-r+1} - 1)/(i-1))$ and $(j = i^{m-r})$

$P_n \equiv (n = m - n)$ and $(k = (i^{n+1} - 1)/(i-1))$ and $(j = i^{n})$

Vậy :

WP(W,Q) $\equiv \exists(r : r >= 0 : (n = m - r)$ and $(k = (i^{n+1} - 1)/(i-1))$ and $(j = i^{n})$

$\equiv (n <= m)$ and $(k = (i^{n}+1 - 1)/(i-1))$ and $(j = i^{n})$

b) Mối liên hệ giữa toán tử lặp và tiên đề lệnh lặp của hệ luật Hoare .

Ta tách việc khảo sát tính đúng một vòng lặp thành hai phần :

+ Phần quan sát sự biến đổi của vòng lặp để dẫn tới khẳng định nó dừng (tính dừng).

+ Phần quan sát sự bất biến của vòng lặp để chứng minh kết quả cuối cùng của nó (đcđk)

Với ý tưởng đó ta tách WP(W,Q) (với W là while do S) thành các thành phần tương ứng và khảo sát mối quan hệ giữa WP(W,Q) và các khẳng định của hệ luật Hoare.

α) Với lệnh bất kỳ S, điều kiện yếu nhất để đảm bảo S dừng là không ràng buộc gì sau khi dừng. Tức là WP(S,true) là tân từ mô tả tập hợp tất cả các trạng thái mà xuất phát từ đó thì bảo đảm S dừng.

Ta có : WP(W,true) $\equiv \exists(k : k >= 0 : P_k)$

Với $P_o \equiv$ (not B) and true $\equiv$ (not B)

$P_k \equiv$ B and WP(S,P_{k-1}) với $k > 0$

(P_o là điều kiện để không thực hiện S lần nào, P_1 là điều kiện để thực hiện S đúng một lần , P_k là điều kiện để thực hiện S đúng k lần.

Ví dụ :

```
W ≡ while ( n <> m) do begin
                          j := j* i ; k := k+j ; n := n+1 ;
                       end ;
```

Ta tính điều kiện đầu để W dừng như sau :

$P_o \equiv$ not (n <> m) $\equiv$ (n = m)

$P_1 \equiv$ B and WP(S,P_o) $\equiv$ (n <> m) and (n+1 = m) $\equiv$ (n+1 = m)

Giả thiết quy nạp rằng $P_k \equiv (n+k = m)$.

Ta có :

Gỉa thiết đúng với k = 0 vì $P_o \equiv (n = m) \equiv (n + 0 = m)$

Gỉa sử gỉa thiết đã đúng với k . Tức là : $P_k \equiv ((n+k) = m)$

Chứng minh gỉa thiết đúng với k+1. Thực vậy:

$P_{k+1} \equiv$ B and WP(S,P_k) $\equiv$ (n <> m) and ((n+1)+k = m) $\equiv$ (n+(k+1) = m)

(WP(S,P_k) $\equiv$ WP (j := j* i ; k := k+j ; n := n+1 ,((n + k) = m)) = (n + (k +1)) = m)

Vậy : $P_i \equiv (n+i = m)$

Tức là : WP(W,true) $\equiv \exists$(i: i>=0; n+i =m) $\equiv$ (n >= m)

β) Quan hệ giữa { P } S { Q } và WP(S,Q)

Theo định nghĩa về tính đúng và WP (S , Q) ta có : S đúng có điều kiện dựa trên điều kiện đầu P và điều kiện cuối Q (đặc tả {P} S {Q} đcđk) nếu và chỉ nếu hội (and) của P và điều kiện yếu nhất bảo đảm sự dừng của S mạnh hơn điều kiện yếu nhất bảo đảm S dừng trong một trạng thái thoả tân từ Q.

Tức là : {P} S {Q} thỏa cđk khi và chỉ khi P and WP(S,true) ==> WP(S,Q)

Như vậy :

{ I and B } S { I } thỏa có đk khi và chỉ khi I and B and WP(S,true) ==> WP(S,I)

{I} while B do S {I and not B} thỏa có đk khi và chỉ khi

{I} and WP(while B do S , true) ==> WP(W, I and not B)

Như vậy chứng minh S giữ bất biến I chính là chứng minh

I and B and wp(W,true) ==> wp(S, I)

Chứng minh W dừng ứng với đkđ P chính là chứng minh : P ==> WP(W,true)

γ) Định lý bất biến cơ sở (Fundamental invariance theorem) của Dijkstra phát biểu một dạng khác của luật về vòng lặp của Hoare .

Định lý: Giả sử I and B and WP(S,true) ==> WP(S,I) (I là bất bất biến của vòng lặp)

thì : I and WP(W,true) ==> WP(while B do S , I and notB)

({I} while B do S {I and not B})

Chứng minh : Ta sẽ chứng minh bằng quy nạp trên k rằng

I and P_k (true) ==> P_k(I and not B) (a)

với : $P_o(Q)$ $\equiv$ not B and Q

$P_k(Q)$ $\equiv$ B and wp(S, $P_{k-1}(Q)$)

Chú ý là $P_k(Q)$ là đkđ yếu nhất bảo đảm vòng lặp while B do S dừng sau đúng k lần lặp trong một trạng thái thoả mãn Q.

(i) Cơ sở I and P_o(true) $\equiv$ I and (not B and true) (định nghĩa)

$\equiv$ not B and (I and not B)

$\equiv$ P_o(I and not B)

(ii) Bước quy nạp : Giả sử (a) đã đúng với k . Tức là :

I and P_k(true) ==> P_k(I and not B)

Ta chứng minh (a) đúng với k+1 .

Thực vậy : I and P_{k+1}(true) $\equiv$ I and B and WP(S,P_k(true)) (định nghĩa)

$\equiv$ B and I and B and WP(S,P_k(true))

$\equiv$ B and I and B and WP(S,true) and WP(S,P_k(true))

(vì WP(S,P_k(true)) $\equiv$ WP(S,true) and WP(S,P_k(true)))

$\equiv$ B and (I and B and WP(S,true)) and WP(S,P_k(true))

==> B and WP(S,I) and WP(S,P_k(true))

(I and B and WP(S,true) ==> WP(S , I) gỉa thiết I là bất biến)

≡ B and WP(S,I and P_k(true)) (phép phân phối _and)

==> B and WP(S,P_k(I and not B))

(vì : I and P_k(true) ==> P_k(I and not B giả thiết quy nạp và tính chất phép phân phối ==>)

≡ P_{k+1}(I and not B)

Tức là: I and P_k(true) ==> P_{k+1}(I and not B)

Theo nguyên lý quy nạp ta suy ra :

I and P_k(true) ==> P_k(I and not B) với mọi k >= 0

Từ điều này ta có :

I and WP(W, true) ≡ I and (k : k >= 0 : P_k(true))

≡ (k : k >= 0 : I and P_k(true))

==> (k : k >= 0 : P_k(I and not B)) ≡ WP(W,I and not B)

Ta có đpcm.

IV. LƯỢC ĐỒ KIỂM CHỨNG HỢP LÝ VÀ CÁC ĐIỀU KIỆN CẦN KIỂM CHỨNG.

1. Lược đồ kiểm chứng.

Để chứng minh tính đúng của đặc tả đoạn chương trình người ta thường :

- Thiết lập các khẳng định về trạng thái chương trình ở các điểm trung gian cần thiết.
- Chứng minh tính đúng của các khẳng định đó.

Những khẳng định về trạng thái chương trình ở những điểm trung gian không chỉ nhằm phục vụ việc kiểm chứng mà còn có mục tiêu là giúp người sử dụng chương trình hiểu được ngữ nghĩa của đoạn chương trình . Tức là góp phần xây dựng một chương trình có dạng thức tốt (dễ đọc, dễ hiểu). Nghệ thuật của việc chứng minh tính đúng của chương trình và xây dựng sưu liệu cho chương trình (ở đây là đưa ra những ghi chú vào chương trình) là ở chỗ làm sao chèn vào các khẳng định trung gian vừa đủ : quá nhiều sẽ làm khó đọc, mất nhiều thời gian kiểm tra, còn quá ít thì không đủ đặc tả ngữ nghĩa .

Trong phần này ta thảo luận ý tưởng chứng minh tính đúng của đoạn chương trình trong dạng lược đồ kiểm chứng tính đúng (Proof tableaux).

Các khái niệm.

- Lược đồ kiểm chứng tính đúng (lđkc) của một đoạn chương trình là một dãy đan xen giữa các khẳng định (assertion) và lệnh (statement) của đoạn chương trình, với bắt đầu và kết thúc bởi các khẳng định.

- Một lược đồ kiểm chứng là đúng (valid) nếu khi ta bỏ đi các khẳng định trung gian thì nó trở thành một đặc tả đúng. Từ những kiến thức đã trình bày ở các phần trên ta suy ra: Một lược đồ kiểm chứng là đúng khi và chỉ khi :

+ Mọi bộ đặc tả dạng {P} S {Q} xuất hiện trong lđkc đều là những đặc tả đúng.

+ Mọi cặp khẳng định đứng liền nhau dạng {H} {T} trong lđkc thì đều thỏa quan hệ P ==> Q đúng.

Từ định nghĩa trên ta thấy : một lđkc có thể biến dạng theo nhiều mức chi tiết. Từ một bộ ba đặc tả gồm : đoạn lệnh S , tân từ mô tả điều kiện đầu P , tân từ mô tả điều kiện cuối Q (đặc tả {P} S {Q}) ta có thể xây dựng nhiều dạng lđkc khác nhau bằng các cách chèn khác nhau các khẳng định trung gian .

Dạng thô nhất của lđkc chính là đặc tả tính đúng của đoạn chương trình nó chỉ chứa 2 khẳng định : một ở đầu đoạn chương trình và một ở cuối đoạn chương trình .

Dạng min nhất của lđkc là lđkc mà mọi lệnh đều bị kèm giữa hai khẳng định (đặc tả ngữ nghĩa tới từng câu lệnh) nó là lược đồ kiểm chứng ở mức chi tiết nhất (lược đồ kiểm chứng chi tiết - lđkcct).

Trung gian giữa hai dạng lđkc trên người ta thường sử dụng lđkc chỉ có các khẳng định trung gian ở những chỗ cần thiết (những chổ quan trọng , những chổ ngoặt trong nội dung ngữ nghĩa của đoạn chương trình).

2. Kiểm chứng tính đúng.

a) Ý tưởng

Để kiểm chứng tính đúng đặc tả của đoạn chương trình S . Tức là khẳng định đặc tả {P} S {Q} đúng . Ta cần thực hiện các việc sau:

+ Xây dựng lđkc hợp lý xuất phát từ đặc tả của đoạn chương trình .

+ Chứng minh tính đúng của lđkc vừa xây dựng .

Trong 2 công việc trên thì việc xây dựng lđkc hợp lý là việc tốn nhiều thời gian và công sức . Việc xây dựng lược đồ chưng minh hợp lý sẻ khác nhau phụ thuộc vào cấu trúc của đoạn lệnh S song thường được tiến hành theo 2 bước sau :

Bước 1 : Từ đặc tả xây dựng lược đồ trung gian (chi tiết hay gần chi tiết) dựa vào các tiên đề (của hệ Hoare hoặc của hệ Dijkstra) mô tả ngữ nghĩa của từng lệnh bằng cách chèn vào các khẳng định trung gian .

Bước 2 : Từ dựng lược đồ trung gian (chi tiết hay gần chi tiết) dựa vào các tiên đề (của hệ Hoare hoặc của hệ Dijkstra) mô tả ngữ nghĩa của từng lệnh bỏ bớt các khẳng đinh trung gian tầm thường (các khẳng định ở những vị trí không quan trong , các khẳng định mà tính đúng của chúng là rõ ràng và dang thức của chúng đơn giản dễ dàng khôi phục lại khi cần) . Giữ lại khẳng định trung gian nào trong lđkc hợp lý là một trong những nghệ thuật của người kiểm chứng nó phản ánh rõ nét mức trí tuệ (khả năng tư duy, kiến thức tích lũy) của người kiểm chứng .

Việc chứng minh tính đúng đầy đủ của lđkc phụ thuộc vào cấu trúc đoạn lệnh S và hệ tiên đề mà ta đã sử dụng để xây dựng lược đồ kiểm chứng hợp lý.

- Trương hợp 1 : Nếu đoạn lệnh S không chứa một lệnh lặp nào cả thì tính dừng được xem là hiển nhiên, khi đó 2 hệ tiên đề là hoàn toàn tương đương .

- Trường hợp 2 : Nếu đoạn lệnh S có chứa lệnh lặp thì tính dừng không phải bao giờ cũng được thỏa nên ta cần phải chỉ ra . Khi đó 2 hệ tiên đề là không tương đương .

+ Nếu trong suốt qúa trình xây dựng lược đồ kiểm chứng ta chỉ sử dụng hệ tiên đề Dijikstra thì không phải kiểm chứng lại tính dừng nữa .

+ Nếu trong qúa trình xây dựng lược đồ kiểm chứng ta có sử dụng (dù chỉ một lần) tiên đề của hệ Hoare thì phải kiểm chứng lại tính dừng (vì tiên đề Hoare không bảo đảm tính dừng) .

b) Kiểm chứng tính đúng đặc tả {P} S {Q} khi S là một dãy lệnh tuần tự.

($S \equiv \{ S_1 ; S_2 ; \ldots ; S_n \}$)

Kiểm chứng tính đúng đặc tả : { P } $S_1 ; S_2 ; \ldots ; S_n$ { Q }

Ví dụ :

Kiểm chứng đặc tả :

{even(k) and (0 < k) and ($y*z^k = x^n$)} (1)
k := k div 2 ;
z := z*z ;
{(0 <= k) and ($y * z^k$) = x^n)} (2)

Bài giải :

Cách 1 : Xây dựng lđkc hợp lý dựa vào hệ Haore .

- Bước 1 : Xây dựng lược đồ kiểm chứng hợp lý.

+ Xây dựng lược đồ kiểm chứng chi tiết :

Từ (1) ta suy ra :

{even(k) and (0 < k) and ($y * z^k$) = x^n)} (2a)
{(0 <= k div 2) and ($y * (z*z)^k$ div 2 = x^n) } (2d)
k := k div 2 ; (2)
{(0 <= k) and ($y * (z*z)^k = x^n$) } (2c)
z := z*z ;
{(0 <= k) and ($y * z^k = x^n$) } (2b)

Diễn giải : Từ (2b) và lệnh gán z := z*z dùng tiên đề gán ta suy ra (2c)
Từ (2c) và lệnh gán k := k div 2 dùng tiên đề gán ta suy ra (2d)

+ Xây dựng lđkc hợp lý từ lđkc chi tiết :

Từ (2) ta suy ra :

{even(k) and (0 < k) and ($y * z^k$) = x^n)} (2a)
{(0 <= k div 2) and ($y * (z*z)^k$ div 2 = x^n) } (2d)
k := k div 2 ; (3)

z := z*z ;

$\{(0 <= k) \text{ and } (y * z^k = x^n)\}$ (2b)

Diễn giải : Từ (2b),(2c),(2d) và 2 lệnh gán tuần tự z := z*z ; k := k div 2 dùng tiên đề tuần tự ta bỏ đi (2c) .

- Bước 2 : chứng minh lđkc hợp lý (3) đúng :

$\{(0 <= k \text{ div } 2) \text{ and } (y * (z*z)^k \text{ div } 2 = x^n)\}$ (2d)

k := k div 2 ; (3a)

z := z*z ;

$\{(0 <= k) \text{ and } (y * z^k = x^n)\}$ (2b)

Ta có : Tính đúng của (3a) được khẳng định dựa vào cách xây dựng .

Kiểm chứng hai khẳng định đi liền nhau :

$\{ \text{even}(k) \text{ and } (0 < k) \text{ and } (y * z^k) = x^n)\} \{(0 <= k \text{ div } 2) \text{ and } (y*(z*z)^{k \text{ div } 2} = x^n)\}$

có quan hệ hàm ý (==>) (hiển nhiên) (3b).

Từ (3a),(3b) áp dụng luật hệ quả ta suy ra (3) đúng .

Nhân xét :

Ta có thể hình thức hóa quá trình chứng minh bằng cách đưa vào ký hiệu :

$I(z,k) \equiv (0 <= k) \text{ and } (y*z^k = x^n)$

Khi đó (2) có thể viết thành :

{even(k) and (0<k) and I(z,k)}

{I(z*z,k div 2)}

k := k div 2 ;

{I(z*z,k)}

z := z*z ;

{I(z,k)}

(3) có thể viết thành :

{even(k) and (0<k) and I(z,k)}

{I(z*z,k div 2)}

k := k div 2 ;

z := z*z ;

{I(z,k)}

Điều kiện cần kiểm chứng là :

even(k) and (0<k) and I(z,k) $\Rightarrow$ I(z*z,k div 2)

Chú ý : Khi có một cặp {P} {Q} xuất hiện trong lược đồ thì khẳng định hàm ý (===>) tương ứng là một điều kiện cần kiểm chứng (đkckc - verification condition). Các điều kiện này là cốt lõi của chứng minh về tđcđk, phần còn lại của chứng minh chỉ là việc áp dụng máy móc các quy luật.

Trong ví dụ trên, đkckc là :

even(k) and (0 < k) and I(z,k)} ==> I(z*z , k div 2)

Đây chỉ là cách nói hình thức của sự kiện là $(z*z)^k$ div 2 = z^k khi k là số nguyên chẵn.

Cách 2 : Xây dựng lđkc hợp lý dựa vào hệ Dijkstra.

Bước 1 : Xây dựng lđkc hợp lý.

- Tính WP(k := k div 2 ; z := z*z , I(z,k))

Ta có : WP(k := k div 2 ; z := z*z , I(z,k))

≡ WP(k := k div 2 ,WP(z := z*z , I(z,k))

≡ WP(k := k div 2 , I(z*z,k)) ≡ I(z * z , k div 2))

+ Chèn WP(k := k div 2 ; z := z*z , I(z,k)) vào (1) ta được lđcm hợp lý :

```
{even(k) and (0<k) and I(z,k)}
          {I(z*z,k div 2 )}
               k := k div 2 ;
               z := z*z ;
           {I(z,k)}
```

Bước 2 : Kiểm chứng tính đúng của lđkc hợp lý .

```
Ta có :   {I(z*z,k div 2 )}
                k := k div 2 ;
                z := z*z ;
               {I(z,k)}       (a) đúng
```

even(k) and (0<k) and I(z,k) ⇒ I(z*z,k div 2) (b) đúng.

Từ (a) , (b) ta suy ra đặc tả đúng

c) Kiểm chứng khi đoạn chương trình có chứa câu lệnh điều kiện

{P} if B then S1 else S2 {Q}

Khi đó ta thêm các khẳng định trung gian dạng:

```
{P}  if B   then  {P and B}       S1   {Q}
            else  {P and not B} S2   {Q}
```

(hoặc :

```
{P}  if B then    {P and B}  S  {Q}
          else    {P and not B}  {Q}
```

khi không có phần else)

vào nơi có lệnh điều kiện tương ứng ta có lđkc trung gian thích hợp .

Ví dụ : Kiểm chứng đoạn chương trình :

{(0 < k) and (y * z^k = x^n}

```
  if even(k)   then  begin
                        k := k div 2 ;
                         z := z*z ;
```

```
                        end
                  else  begin
                           k :=  k -1 ;
                           y :=  y*z  ;
                        end
```

{(0 <= k) and (y * z^k = x^n)}

Cách 1: Dùng hệ tiên đề Hoare.

+ Bước 1 : Xây dựng lđkc hợp lý.

Đặt I(y,z,k) ≡ (0 <= k) and ($y*z^k = x^n$)

Đặc tả có dạng : {(0 < k) and I(y,z,k)}

```
if  even(k)   then  begin
                       k :=  k  div  2 ;
                       z :=  z*z  ;
                    end
              else  begin
                       k :=  k -1 ;
                       y :=  y*z  ;
                    end
{I(y,z,k)}
```

Chèn các khẳng định trung gian (dựa vào tiên đề gán của Hoare)

```
{(0 < k ) and I(y,z,k)}
if  even(k)   then  {even(k) and (0 < k ) and I(y,z,k)}
                     begin
                { I(y ,z*z , k div 2 ) }
                       k :=  k  div  2 ;
                  { I( y , z*z , k ) }
                       z :=  z*z  ;
                        end;
                  {I(y,z,k)}
              else
      {(not even(k)) and (0 < k ) and I(y,z,k)}
                     begin
               { I(y*z ,z , k –1) }
                       k :=  k -1 ;
                  { I(y*z , z , k ) }
                  y :=  y*z  ;
                    end
                  {I(y,z,k)}
```

Bỏ đi các khẳng định trung gian tầm thường (dựa vào luật tuần tự của Haore) ta có lđkc hợp lý : {(0 < k) and I(y,z,k)}

```
if  even(k)   then    {even(k) and (0 < k ) and I(y,z,k)}
```

{ I(y ,z*z , k div 2) }
begin k := k div 2 ;
z := z*z ;
end
{I(y,z,k)}
else
{(not even(k)) and (0 < k) and I(y,z,k)}
{ I(y*z ,z , k –1) }
k := k -1 ;
y := y*z ;
{I(y,z,k)}

+ Bước 2: Chứng minh lược đồ kc đúng.

Các cặp khẳng định đứng liền nhau xuất hiện trong lược đồ :

{even(k) and (0 < k) and I(y,z,k)} { I(y ,z*z , k div 2) } (a)

và {(not even(k)) and (0 < k) and I(y,z,k)} { I(y*z ,z , k –1) }(b)

Các hàm ý tương ứng cần phải chứng minh đúng :

{even(k) and (0 < k) and I(y,z,k)} ==> { I(y ,z*z , k div 2) } (a*) (kiểm chứng ?)

và {(not even(k)) and (0 < k) and I(y,z,k)} ==> { I(y*z ,z , k –1) }(b*) (Kiểm chứng ?)

Từ (a*) và (b*) ta suy ra điều phải kiểm chứng.

Cách 2: Dùng hệ tiên đề Dijkstra.

- Bước 1 : Xây dựng lđkc hợp lý.

Đặt I(y,z,k) ≡ (0 <= k) and ($y*z^k = x^n$)

S_1 ≡ begin k := k div 2 ; z := z*z ; end ;

S_2 ≡ begin k := k -1 ; y := y*z ; end ;

B ≡ even(k)

Đặc tả có dạng : {(0 < k) and I(y,z,k)}
if B then S_1 else S_2
{I(y,z,k)}

+ Tính WP(if B then S_1 else S_2 , I(y,z,k))
≡ (B ==> WP(S_1 , I(y,z,k))) and (not B ==> WP(S_2 , I(y,z,k)))
≡ (dành cho người đọc)

+ Chèn khẳng định WP(if B then S_1 else S_2 , I(y,z,k)) vào đặc tả theo tiên đề chọn của Dijkstra ta được lđkc hợp lý dạng :

{(0 < k) and I(y,z,k)}
{ WP(if B then S_1 else S_2 , I(y,z,k)) }
if B then S_1 else S_2
{I(y,z,k)}

- Bước 2 : Chứng minh lđkc đúng.

Cặp khẳng định đứng liền nhau trong lđkc là :

{(0 < k) and I(y,z,k)} { WP(if B then S_1 else S_2 , I(y,z,k)) }

Ta cần chứng minh tính đúng của hàm ý tương ứng :

{(0 < k) and I(y,z,k)} ==> { WP(if B then S_1 else S_2 , I(y,z,k)) } (*)

(CM * danh cho người đọc)

Từ (*) suy ra điều phải kiểm chứng.

d) Kiểm chưng khi đoạn chương trình có chứa lệnh lặp while B do S.

Cách thứ 1: Sử dụng hệ tiên đề Dijkstra.

- Bươc 1 : Xây dựng WP(While B do S) và chèn vào trước lệnh lặp .

. . . . . . { WP(while B do S } while B do S . . . . . .

- Bước 2: Xây dựng lđkc hợp lý từ lđkc trên.
- Bước 3 : Chứng minh tính đúng của các điều kiện cần kiểm chứng.

Cách thứ 2 : Sử dụng hệ tiên đề Hoare.

- Bươc 1 : Phát hiện bất biến I của vòng lặp và chèn các khẳng định trung gian tương ứng vào trước giữa và sau lệnh lặp (tiên đề Haore) .

```
{(Invariant) I}
while B do
   {I and B} S {I}
   {I and not B }
```

- Bước 2. Xây dựng lđkc hợp lý từ lđkc trên.
- Bước 3 : Chứng minh tính đúng của các điều kiện cần kiểm chứng .

Bước 4 : Chứng minh lệnh lặp dừng .

Ví dụ : Kiểm chứng đặc tả :

```
      { 0 <= n }
  y := 1 ; z := x ; k := n ;
      while (0 <> k ) do
            begin
              k := k -1 ; y := y*z
            end
        {y = x^n }
```

Biết bất biến của vòng lặp : I(y,z,k) ≡ (k >= 0) and ($y*z = x^n$)

Bài giải theo cách 1:

Dựa vào hệ Hoare ta xây dựng lđkc chi tiết xuất phát từ điều kiện đầu , điều kiện cuối và bất biến .

{0 <= n}

{I(1,x,n)}

y := 1 ;

{I(y,x,n)}

z := x ;

{I(y,z,n)}

```
          k := n ;
        { I(y,z,k)}
          while (0 <> k )  do    begin
                      {I(y,z,k) and (k <> 0 )}
                      {I(y*z,z,k-1)}
                         k := k -1 ;
                     {I(y*z,z,k)}
                       y :=  y*z ;
                    {I(y,z,k)}
                         end
        {I(y,z,k) and (k = 0 )}
        {y = x^n }
```

Bỏ đi các khăng định trung gian tầm thường ta có lđcm hợp lý dạng :

```
        {0 <= n}
         {I(1,x,n)}
           y := 1 ;
          z := x ;
           k := n ;
        { I(y,z,k)}
          while (0 <> k )  do
                           begin
                      {I(y,z,k) and (k <> 0 )}
                      {I(y*z,z,k-1)}
                         k := k -1 ;
                          y :=  y*z ;
                       {I(y,z,k)}
                           end
                   {I(y,z,k) and (k = 0 )}
                    {y = x^n }
```

Các điều kiện cần kiểm chứng là :

(0 <= n) ==> I(1,x,n)

I(y,z,k) and (k = 0) ==> $y = x^n$.

I(y,z,k) and (k <> 0) ==> I(y*x,z,k-1)

Thay I(y,z,k) ≡ (0 <= k) and ($y * z^k = x^n$) ba đkckc trên sẽ trở thành :

(Phần chuẩn bị) (0 <= n) ==> ($1*x^n = x^n$)and (0 <= n) (hiển nhiên)

(Phần kết thúc lặp) ($y * z^k = x^n$) and (0 <= k) and (k = 0)==> $y = x^n$ (hiển nhiên)

(Phần thân vòng lặp) ($y * z^k = x^n$) and (0 <= k) and (k <> 0)

==> ($(y*z)*z^{k-1} = x^n$) and (0 <= k -1)

≡ ($y * z^k = x^n$) and (0 < k)

==> ($y*z^k = x^n$) and (0 <= k -1) (hiển nhiên)

3. Tập tối tiểu các điều kiện cần kiểm chứng.

Một lđkc đầy đủ trong đó mỗi lệnh đều được kèm giữa hai khẳng định rõ ràng là chi tiết quá mức. Thực ra sử dụng tri thức của ta về các đkđ yếu nhất của những lệnh khác lệnh lặp, ta có thể mô tả một giải thuật để sản sinh ra một chứng minh hoàn chỉnh theo kiểu Hoare về tính đúng có điều kiện của đoạn lệnh S dựa trên điều kiện đầu P và điều kiện cuối Q, với giả định là mỗi vòng lặp while trong S được cung cấp kèm theo bất biến của nó.

Về nguyên tắc, một bộ chứng minh định lý tự động (theorem prover), với khả năng kiểm chứng các điều kiện có dạng P ==> R có thể được dùng để chứng minh một cách tự động tđcđk của 1 đoạn chương trình . Điểm quan trọng mà ta rút ra từ các phần đã trình bày là: phần cốt lõi trong một chứng minh về tđcđk là việc phát hiện ra các bất biến và sau đó việc kiểm chứng các điều kiện hàm ý nhằm sử dụng luật hệ quả.

Chúng ta không mô tả giải thuật để sản sinh các chứng minh kiểu Hoare, thay vào đó, ta sẽ trừu tượng hoá từ nó quá trình sản sinh ra tập hợp các điều kiện cần kiểm chứng.

Xét một đoạn CT bất kỳ với các đkđ P và đkc Q. Ta sẽ xây dựng từ P, S và Q bằng quy nạp một điều kiện đầu yếu nhất dựa vào S và Q, ký hiệu là pre(S,Q), và hai tập hợp các điều kiện cần kiểm chứng V'(S,Q) and V(P,S,Q) như sau :

1. Nếu S là lệnh gán x := bt thì pre(S,Q) là WP(S,Q) và V'(S,Q) rỗng.

Tức là : pre(x := bt ,Q(x))) $\equiv$ WP(x := bt ,Q(x)) $\equiv$ Q(bt) và V'(x := bt ,Q) $\equiv \varnothing$.

2. Nếu S có dạng S_1 ; S_2 thì pre(S,Q) là pre(S_1, pre(S_2,Q)) và V'(S,Q) là hội của

V'(S_2 , Q) và V'(S_1 ,pre(S_2 ,Q)).

Tức là : pre(S_1; S_2 , Q) $\equiv$ pre(S_1, pre(S_2,Q))

Và V'(S_1; S_2 ,Q) $\equiv$ V'(S_2 , Q) $\cup$ V'(S_1 ,pre(S_2 ,Q)).

3. Nếu S có dạng if B then S_1 else S_2 thì pre(S,Q) là :

(B and pre(S_1,Q)) or (not B and pre(S_2,Q)) và V'(S,Q) là hội của V'(S_1,Q) và V'(S_2,Q).

Tức là : pre(if B then S_1 else S_2 ,Q) $\equiv$ (B and pre(S_1 ,Q)) or (not B and pre(S_2 ,Q)) Va V'(if B then S_1 else S_2 ,Q) $\equiv$ V'(S_1,Q) $\cup$ V'(S_2,Q).

4. Nếu S có dạng while B do S_1 và I là bất biến của vòng lặp thì pre(S,Q) là I, và V'(S ,Q) là hội của V(I and B , S_1 ,I) và tập hợp chỉ gồm một điều kiện I and not B ==> Q.

Tức là : pre(S,Q) $\equiv$ I và V'(S ,Q) $\equiv$ V(I and B , S_1 ,I) $\cup$ { I and not B ==> Q }.

5. Trong mọi trường hợp V(P,S,Q) là hội của V'(S,Q) và tập hợp chỉ gồm một điều kiện P ==> pre(S,Q).

Tức là : V(P,S,Q) $\equiv$ V'(S,Q) $\cup$ { P ==> pre(S,Q) }.

Các chức năng của pre(S,Q) , V'(S, Q) ,và V(S,P,Q) trong quá trình này được mô tả bởi các mệnh đề sau :

(P1) Nếu mọi đkckc trong tập hợp V(S,Q) đều đúng thì S là đcđk dựa trên đkđ pre(S,Q) và đkc Q. Tức là : { pre(S,Q) } S { Q } đúng có điều kiện.

(P2) Nếu mọi đkckc trong tập hợp V(P,S,Q) đều đúng thì S là đcđk dựa trên đkđ P và đkc Q. Tức là : { P } S {Q } đúng có điều kiện.

Tính chất (P1) có thể được chứng minh bằng quy nạp trên kích thước của S, ở đây kích thước của S có được bằng cách đếm là 1 cho mỗi lần xuất hiện các ký hiệu ':=', ';', 'if', 'while' trong S. Tính chất (P2) là một hệ quả trực tiếp của (P1).

Chú ý rằng pre(S,Q) khác với wp(S,Q) chỉ khi có lệnh while. Điều này xác nhận là trong trường hợp tổng quát, không có khả năng tạo lập một công thức đóng cho đkđ yếu nhất của lệnh while và nhấn mạnh tầm quan trọng của việc ghi nhận những tính chất bất biến trong các sưu liệu chương trình.

Ví dụ 1 : Với đặc tả gồm :

Dẫy lệnh tuần tự S : S ≡ tg := tg + a[k] ; k := k+1 ;

đkc Q ≡ I(k, tg) ≡ (tg = S(i : 0 <= i < k : a[i]))

đkđ P ≡ I(k,s) and (k <> n)

Ta áp dụng các bước 1 và 2 được :

V'(S, I(k, tg)) là rỗng .

pre(S, I(k, g)) là I(k+1, tg+a[k])

tập các đkckc

V(P,S,Q) ≡ V(I(k,tg) and k <> n, S, I(k, tg)) chứa một điều kiện là

I(k,tg) and k <> n ==> I(k+1, tg+a[k])

Tức là : (tg = S(i : 0<= i< k : a[i])) and (k <> n)

==> tg + a[k] = S(i: 0 <= i <= k+1 : a[i]) (1)

Vidụ 2 : Xét đặc tả đoạn chương trình tính tổng các phần tử của một array

```
            {0 <= n}
             k := 0 ; tg := 0 ;
{(Invariant ) I(k,tg) }  ≡ {  tg = S(i: 0<= i <k : a[i])}
    while (k <> n ) do
                   begin
                     tg := tg + a[k] ; k := k+1 ;
                   end
     {tg = S(i: 0 <= i < n : a[i])}
```

Tách đoạn chương trình thanh 2 nhóm :

+ Nhóm lệnh tuần tự : S_o ≡ k := 0 ; tg := 0 ;

+ Lệnh while : W ≡ while k <> n do begin
tg := tg + a[k] ; k := k+ 1
end

Theo quy tắc 2, ta cần tính pre(W,Q) và V'(W,Q) với

Q ≡ tg = S(i: 0 <= i < n : a[i])

Bây giờ, dùng quy tắc 4,

pre(W,Q) ≡ I(k,tg) ≡ tg = S(i : 0 <= i < k : a[i])

Cũng vậy V'(W,Q) bao gồm V(I(k,tg) and k <> n, S_1, I(k,tg)) với S_1 là nhóm lệnh trong vòng lặp, và điều kiện

I(k,tg) and (k = n)==> tg = S(i : 0<= i <n : a[i]) (2)

Cuối cùng, ta có thể tìm được

pre(S_o, I(k,tg)) ≡ 0 = S(i: 0 <= i <0 : a[i])

và tập hợp các đkckc cho So bao gồm chỉ một điều kiện

(0 <= n) ==> pre(S_o, I(k,tg)) (3)

Như vậy, có 3 đkckc cho CT, đó là các điều kiện (1), (2), (3).

PHU LỤC
MỘT SỐ KIẾN THỨC VỀ LOGIC

I. LOGIC TOÁN.

Trong đời sống hàng ngày, người ta cần có những lý luận để từ các điều kiện được biết hay được giả định (các tiền đề - premises) có thể suy ra các kết luận (conclusion) đúng. Hãy xét 2 lý luận sau :

Lý luận (1) : - Các tiền đề :
+ Nếu hôm nay trời đẹp thì tôi đi chơi.
+ Nếu tôi đi chơi thì hôm nay về trễ .
- Gỉa thiết : Hôm nay trời đẹp .
- kết luận : Hôm nay tôi sẽ về trễ .

Lý luận (2) : - Các tiên đề :
+ Nếu hôm nay rạp hát không đóng cửa thi tôi sẽ xem phim.
+ Nếu tôi xem phim thì tôi sẽ không soạn kịp bài .
- Gỉa thiết : Hôm nay rạp hát không đóng cửa .
- kết luận : Hôm nay tôi sẽ không soạn kịp bài.

Hai lý luận trên là đúng và có cùng dạng lý luận. Chúng đúng vì có dạng lý luận đúng, bất kể ý nghĩa mà chúng đề cập đến.

Còn lý luận sau :

Lý luận (3) : - Các tiền đề :
+ Nếu trời đẹp thì tôi đi chơi.
+ Nếu tôi đi chơi thì tôi sẽ về trễ.
- Giả thiết : Hôm nay tôi về trễ.
- kết luận : Hôm nay trời đẹp .

là lý luận sai và mọi lý luận dạng như vậy đều sai .

Logic toán học quan tâm đến việc phân tích các câu (sentences), các mệnh đề (propositions) và chứng minh với sự chú ý đến dạng (form) lược bỏ đi sự việc cụ thể.

II. LOGIC MỆNH ĐỀ.

1. Phân tích

Phân tích lý luận (1) ta thấy nó sử dụng các mệnh đề cơ sở sau :

. Hôm nay trời đẹp
. Tôi đi chơi
. Tôi sẽ về trễ.

Mỗi mệnh đề (proposition) là một phát biểu đúng (true) hay sai (false).

Biểu thị tượng trưng lần lượt các mệnh đề trên bởi các tên A, B, C, ta ghi lại dạng lý luận của (1) như sau :

. Nếu A thì B (4)

. nếu B thì C

Có A kết luận được : C

Đây cũng là dạng lý luận của (2) .

Thường một phát biểu sẽ gồm nhiều phát biểu nhỏ nối kết với nhau bằng các liên từ "và" , "hay" , "vì vậy " ,"kết quả là" ...

Một mệnh đề đơn (simple proposition) là mệnh đề không chứa mệnh đề khác.

Một mệnh đề phức (compound proposition) là mệnh đề được tạo thành từ hai hay nhiều mệnh đề đơn .Việc nối kết này được thực hiện bởi các liên từ logic.

2. Các liên từ logic.

ký hiệu	ý nghĩa là
and	và
or	hay
not	không
==>	nếu...thì...
<==>	...nếu và chỉ nếu...

Với các ký hiệu này, (4) có thể được viết như sau:

[(A ==> B) and (B ==> C) and A] ====> C

Nếu A thì B và nếu B thì C và A Thì suy ra C

Tức là mệnh đề phức hợp [(A ==> B) and (B ==> C) and A] ==> C .

Nói chung một lý luận sẽ được chuyển thành một mệnh đề phức với dạng :

[(tiên đề 1) and (tiên đề 2) and ...] ====> kết luận .

3. Ýnghĩa của các liên từ Logic. Bảng chân trị.

Các liên từ nối kết các mệnh đề thành phần tạo nên mệnh đề mới, mà tính đúng sai của nó được xác định từ tính đúng sai của các mệnh đề thành phần theo qui luật được khái quát trong các bảng giá trị đúng sai sau đây :

p	q	p and q	p or q	p ==> q	p <==> q
F	**F**	**F**	**F**	**T**	**T**
F	**T**	**F**	**T**	**T**	**F**
T	**F**	**F**	**T**	**F**	**F**
T	**T**	**T**	**T**	**T**	**T**

P	not P
T	F
F	T

4. Lý luận đúng.

Một lý luận có thể được biểu diễn bởi một mệnh đề phức trong đó các tiên đề được nối kết với nhau bằng liên từ and và các tiên đề nối kết với kết luận bằng liên từ ==>

Định nghĩa : Một lý luận là đúng (valid) nếu và chỉ nếu với mọi bộ giá trị (đúng, sai) có thể của các mệnh đề thành phần, nó luôn luôn đúng (true)

Ví dụ 1: Lý luận (4) đúng vì với mọi khả năng của A,B,C mệnh đề :

[(A ==> B) and (B ==> C) and A] ==> C đều có gía trị đúng.

Bảng chân trị sau khẳng định điều đó:

A	B	C	[(A ==> B) and (B ==> C) and A] ==> C
F	**F**	**F**	**[T and T and F] ==> F (T)**
F	**F**	**T**	**[T and T and F] ==> T (T)**
F	**T**	**F**	**[T and F and F] ==> F (T)**
F	**T**	**T**	**[T and T and F] ==> T (T)**
T	**F**	**F**	**[F and T and T] ==> F (T)**
T	**F**	**T**	**[F and T and T] ==> T (T)**
T	**T**	**F**	**[T and F and T] ==> F (T)**
T	**T**	**T**	**[T and T and T] ==> T (T)**

Ví dụ 2: Lý luận (3) là sai .

Đặt : A : hôm nay trời đẹp

B : Tôi đi chơi

C : Tôi về trễ

Dạng lý luận (3) là : [(A ==> B) and (B ==> C) and C] ==> A là sai vì với A, B False , C true thì mệnh đề :

[(A ==> B) and (B ==> C) and C] ==> A nhận gía trị False

A B C	[(A ==> B) and (B ==> C) and C] ==> A
F F T	[T and T and T] ==> F

5. Tương đương (Equivalence).

a) Định nghĩa:

Hai mệnh đề P và Q được gọi là tương đương nhau (ký hiệu P ≡ Q), nếu mệnh đề P <==> Q luôn nhận giá trị đúng (True) với mọi khả năng đúng sai của các mệnh đề thành phần .

Ta có thể chứng minh một sự tương đương bằng cách lập bảng chân trị .

Ví dụ: chứng minh : p and q ≡ not(not p or not q).

Bảng chân trị :

p	q	p and q	not (not p or not q)
F	F	F	not (T or T)
F	T	F	not (T or F)
T	F	F	not (F or T)
T	T	T	not (F or F)

b) Một số tương đương hữu ích.

(hãy chứng minh chúng bằng cách lập bảng chân trị)

Các hằng : P or true ≡ true
P or false ≡ p
p and true ≡ p
p and false ≡ false
true ==> p ≡ p
false ==> p ≡ true
p ==> true ≡ true
p ==> false ≡ not p

Luật loại trừ trung gian : p or not p ≡ true
Luật về mâu thuẫn : p and not p ≡ false
Luật phủ định : not not p ≡ p
Luật Kết hợp : p or (q or r) ≡ (p or q) or r
p and (q and r) ≡ (p and q) and r
p <==> (q <==> r) ≡ (p <==> q) <==> r
Luật giao hoán : p and q ≡ q and p
p or q ≡ q or p

p <==> q ≡ q <==> p

luật phân phối : p and (q or r) ≡ (p and q) or (p and r)

p or (q and r) ≡ (p or q) and (p or r)

Luật đồng nhất : p or p ≡ p

p and p ≡ p

Luật De Morgan : not (p or q) ≡ not p and not q

not (p and q) ≡ not p or not q

Luật hàm ý : p ==> q ≡ not p or q

p ==> q ≡ not q ==> p

(p and q) ==> r) ≡ (p ==> (q ==> r))

luật nếu và chỉ nếu : p <==> q ≡ ((p ==> q) and (q ==> p))

p <==> q ≡ ((p ==> q) and (not p ==> not q))

p <==> q ≡ ((p and q) or (not p and not q))

6. Tính thay thế, tính truyền và tính đối xứng.

Khi 2 mệnh đề P và Q là tương đương thì ta có thể thay thế cái này bởi cái kia trong một mệnh đề bất kỳ mà không làm sai trị của nó.

Ta có thể chứng minh tính chất của một mệnh đề bằng cách biến đổi nó thành các mệnh đề tương đương.

Ví dụ: ta chứng minh rằng (p and (p ==> q)) ==> q là một lý luận hợp logic bằng cách biến đổi tương đương.

(p and (p ==> q)) ==> q ≡ (p and (not p or q)) ==> q (hàm ý)
≡ ((p and not p) or (p and q)) ==> q (phân phối)
≡ (false or (p and q)) ==> q (mâu thuẫn)
≡ (p and q) ==> q (hằng)
≡ not (p and q) or q (hàm ý)
≡ (not p or not q) or q (De Morgan)
≡ not p or (not q or q) (kết hợp)
≡ not p or (q or not q) (giao hoán)
≡ not p or true ≡ true

Quan hệ "tương đương" giữa các mệnh đề có tính :

+ Phản xạ : p ≡ p
+ Đối xứng : nếu p ≡ q thì ta cũng có q ≡ p
+ Bắc cầu : nếu p ≡ q và q ≡ r thì ta cũng có p ≡ r.

7. Bài toán suy diễn logic.

Xét bài toán : Trên hòn đảo có hai loại người sinh sống : quân tử và tiểu nhân.

Quân tử luôn nói thật và tiểu nhân luôn nói dối. Một người hỏi một dân cư A trên đảo : "có phải anh là một quân tử ?". A đáp :"nếu tôi là quân tử thì tôi thua tiền anh ". Hãy chứng minh rằng : A nhất định phải thua tiền.

Ta mô hình hóa bài toán như sau :

Đặt các mệnh đề P : A là quân tử. Q : A phải trả tiền.

Kết luận phải chứng minh là Q.

Khảo sát giả thiết của bài toán:

+ Mệnh đề khẳng định : " A là tiểu nhân " là not P

+ A phát biểu một mệnh đề S. giả thiết cho biết : Nếu A là quân tử thì S phải đúng tức là : P ==> S

+ Nếu A là tiểu nhân thì S phải sai : not p ==> not s

+ S là một hàm ý : " Nếu A là quân tử thì A phải trả tiền".

Ta biểu thị S bởi : p ==> q

Như vậy tiền đề là : (P ==> S) and (not P ==> not S)

theo luật tương đương (k) ta có thể viết là : P <==> S.

Bài toán được phát biểu dưới dạng thuần tuý logic như sau :

Cho tiền đề P <==> (P ==> Q)

Có thể suy diễn được kết luận Q không ?

Ta sẽ xác lập rằng (lý luận trên là đúng) mệnh đề (P <==>(p ==> Q)) ==> Q là đúng với mọi bộ giá trị đúng sai của các mệnh đề thành phần .

Có hai cách : (a) Dùng bảng giá trị đúng sai .

P	Q	(P <==> (P ==> Q))	==>	Q
T	T	(T <==> T)	==>	T
F	T	(F <==> T)	==>	T
T	F	(T <==> F)	==>	F
F	F	(F <==> T)	==>	F

(b) Dùng cách thay thế bằng các mệnh đề tương đương .

P <==> (P ==> Q) ≡ P <==> (not P or Q) (hàm ý)

≡ [(P and (not P or Q)] or [not P and not (not P or Q)] (tương đương)

mà not P and not (not P or Q) ≡ not P and (not not P and not Q)

≡ not P and (P and not Q)

≡ (not P and P) and not Q

≡ false and not Q ≡ false

và P and (not P or Q) ≡ (P and not P) or (P and Q)

≡ false or (p and Q)

≡ P and Q

Như vậy P <==> (P ==> Q) ≡ P and Q

Từ đó [P<==>(P ==>Q)] ==> Q ≡ (P and Q) ==> Q

≡ not (P and Q) or Q
≡ (not P or not Q) or Q
≡ not P or (not Q or Q)
≡ not P or true ≡ true

Với các bài toán chỉ liên quan đến ít mệnh đề như trong ví dụ trên, cách dùng bảng chân trị đơn giản hơn . Nhưng nên cố gắng sử dụng cách biến đổi tương đương, bởi vì áp dụng thực tiễn của nó là lớn hơn nhiều.

8. Các luật suy diễn (rules of inference).

Tương tự như bài toán ở mục trên, trong nhiều lĩnh vực, người ta cần phải xuất phát từ một tập hợp các tiền đề, và tìm cách khẳng định một kết luận có phải là hệ quả của các tiền đề đó không ?

Cách giải quyết là người ta xây dựng cho lĩnh vực đó một hệ thống các tiên đề (axioms), tức là các khẳng định được xem như đương nhiên đúng (valid) và một tập hợp các luật suy diễn (rules of inference – tập các qui tắc cho phép xây dựng các khẳng định đúng mới xuất phát từ các tiên đề và các khẳng định đã có) .

Trong khung cảnh này, một khẳng định được thiết lập như vậy được gọi là một định lý . Một chứng minh hình thức (formal proof) là một dãy có thứ tự của các khẳng định, mà mỗi khẳng định hoặc là tiên đề, hoặc được suy diễn từ các khẳng định đi trước, bằng một luật suy diễn nào đó.

a) Hệ luật suy diễn của Gentden cho logic mệnh đề. Trong đó mỗi luật suy diễn sẽ được viết dưới dạng : $\dfrac{S_1, S_2, \ldots, S_n}{S}$

diễn tả nếu đã có các mệnh đề dạng $S_1, S_2, \ldots, S_n$ thì ta có thể suy ra S

Các luật thêm vào	Các định luật loại bỏ
and_I $\dfrac{p, q}{p \text{ and } q}$	**and_E** $\dfrac{p \text{ and } q}{p}$ $\dfrac{p \text{ and } q}{q}$
or_I $\dfrac{p}{p \text{ or } q}$ $\dfrac{q}{p \text{ or } q}$	**or_E** $\dfrac{p \text{ or } q, [p]\, r, [q]\, r}{r}$
==>_I $\dfrac{[p]\ q}{p \Longrightarrow q}$	**==>_E** $\dfrac{p, \ p \Longrightarrow q}{q}$
not_I	**not_E**

[p] false ________ not p **<==>_I** p ==> q , q ==> p ________________ p <==> q	p ,not p false not not p ________ _____ ________ false p p **<==>_E** p <==> q p <==> q ________ ________ p <==> q p <==> q

Các luật được chia làm các luật thêm và các luật loại bỏ : Các luật thêm vào cho phép suy ra một khẳng định mới trong đó có xuất hiện thêm một liên từ logic. Còn các luật loại bỏ thì loại bỏ một liên từ logic.

Luật and_I nói rằng nếu có thể chứng minh được p và q thì ta suy được ra p and q .

Luật and_E nói rằng nếu chứng minh được p and q thì ta suy được từng thành phần p và q .

Luật or_E sử dụng 3 tiền đề : đã có p or q , nếu giả định p đúng thì chứng minh được r , nếu giả định q đúng thì chứng minh được r. khi đó luật này cho phép kết luận r đúng. Đây chính là phân tích theo trường hợp (case analysis) vẫn thường được dùng trong lý luận hàng ngày .

Luật ==>E thường được gọi là modusponens (tam đoạn luận). Nó nói rằng có q nếu chứng minh được p và p ==> q .

Luật not_I nói rằng nếu xuất phát từ giả định p mà có mâu thuẫn thì cho ta kết luận not p . Cùng với luật này , cần bổ sung thêm luật về loại trừ trung gian **true**

p or not p

được phát biểu như tiên đề (tức là luật suy diễn không cần tiền đề).

III. LOGIC TÂN TỪ.

1. Khái niệm

Trong logic mệnh đề , mỗi mệnh đề có giá trị xác định hoặc là T (đúng) hoặc là F (sai) . Trong thực tế người ta hay gặp và cần làm việc với những khẳng định mà tính đúng sai của nó phụ thuộc vào các đối tượng thực sự được thay thế .

Ví dụ xét phát biểu sau : " x là số nguyên tố ".

Gọi mệnh đề này là P(x), đây là một mệnh đề mà tính đúng sai của nó chỉ được xác định hoàn toàn khi ta "thế" một giá trị hằng cho "biến" x.

Ví dụ P(5) là T (dúng) , P(6) là F (sai) .

Trong logic tân từ , người ta phát biểu các mệnh đề bằng cách sử dụng những khái niệm sau:

a) Các hằng: là các đối tượng cụ thể tồn tại trong lĩnh vực mà người ta đang khảo sát .

Ví dụ : + Các hằng số 5,6,10.2,...

+ Các hằng logic T(đúng) , F(sai)

Trong trường hợp tổng quát ,người ta thường đại diện cho các hằng bằng các chữ cái viết thường ở đầu bảng từ vựng: a,b,c...,a_1 ,b_1 , c_1 ,...

b) Các biến (Variable): là các tên tượng trưng . Mỗi biến được ấn định một miền giá trị là tập các đối tượng mà nó có thể nhận.

Ví dụ: + Các biến số nguyên n, j , k ,. . . với các tập trị là các tập con của tập số nguyên Z .

+ Các biến số thực x, y, z, . với các tập trị là các tập con của tập số thực R .

+ Các biến véc tơ V, W, . . . với các tập trị là các tập con của tập tích ĐềCác R X R X R X ... X R (R^n)

Thường dùng các chữ cái viết thường ở cuối bảng từ vựng để biểu thị các biến : x,y,z,...,x_1 ,y_1 ,z_1 ,... Từ dây về sau ,mỗi biến nếu không được nói rõ đều được xem là biến nguyên .

c) Các toán tử (Operotors , hay hàm (functions)) là các ánh xạ từ các tập hợp đối tượng vào các tập hợp đối tượng trong lĩnh vực đang khảo sát. Ta sẽ thường dùng các toán tử toán học sau : + , - , * , / , div , mod

Một toán tử có thể có một hay nhiều toán hạng (ngôi) .

Ví dụ : + Toán tử "đối" (biểu thị bởi -) là một ngôi : -x

+ Toán tử - ,+, - , * , / , div, mod là hai ngôi : 2 + 3 , x * y

d) Các hàm logic hay các tân từ (predicates) . Đó là các ánh xạ từ tập hợp các đối tượng vào tập boolean {true,false}, ta sẽ thường dùng các tân từ là các quan hệ toán học như sau :

+ Các quan hệ so sánh : = , <> , > , >= , < , <=

+ Các quan hệ tập hợp : $\subseteq$, $\supseteq$, . . .

+ Các quan hệ khác : odd(x) kiểm tra xem x có lẻ không ?

even(x) kiểm tra xem x có chẵn không ?

e) Các liên từ logic : đây là các toán tử trên tập boolean mà ta gặp trong logic mệnh đề: and , or , not , ==> , <==>.

f) Các lượng từ phổ dụng $\forall$ và tồn tại $\exists$ (sẽ nói rõ ở mục sau)

Các biến logic , các tân từ trong đó có chứa các hằng hay biến hay hàm được gọi là các công thức cơ sở (formule elementaire)

Ví dụ : Các công thức cơ sở

- Biến logic : hôm-nay-trời-đẹp , tôi-về-lúc-8-giờ ,...
- tân từ : 5 > 2

x > 5

x + 5 > y - 3

Từ các công thức cơ sở này,người ta có thể thành lập các công thức phức hợp (formule complexe) bằng cách nối kết chúng dùng các liên từ logic và các lượng từ . Mỗi công thức phức hợp có thể xem là một tân từ mới.

Ví dụ : Công thức phức hợp

a) Hôm_nay_trời_đẹp and x > y

b) x > y ==> x > z

2. Các lượng từ logic

Ngoài các liên từ logic , người ta có thể tạo ra các công thức phức hợp bằng cách gắn với các công thức các lượng từ logic .

a) Lượng từ phổ dụng.

Để nói rằng mỗi phần tử của một tập đều có tính chất P ta dùng lượng tử phổ dụng ∀ (đọc là với mọi) .

Ví dụ để nói rằng tất cả các phần tử của array a là không âm ta viết :

∀ (i : 0 <= i < n : a [i] >= 0)

Biểu thức này được đọc như sau :

∀ (i	Với mọi (số nguyên) i
: 0 <= i < n	sao cho i nằm giữa 0 và n-1
: a[i] >= 0	thì a [i] là không âm

Dạng chung : ∀ (danh sách biến : R : P)

Với : R là tân từ hạn chế tập hợp được xét trong không gian định bởi danh sách biến , P là tân từ mà mỗi phần tử trong tập được xét phải thoả.

Mệnh đề phổ dụng chỉ đúng khi mọi phần tử trong tập xác định bởi R đều thoả P.

Ví dụ : Cho a là array [0...n-1] of Integer

- Khẳng định : "a [k] là phần tử lớn nhất trong array"

  ∀ (i : 0 <= i < n : a [k] >= a [i])

- Khẳng định : "các phần tử của a tạo thành cấp số cộng b,b+d, b+2d, . .

  ∀ (i : 0 <= i < n : a [i] = b + i*d)

- Khẳng định : "a dược sắp theo thứ tự không đơn giản" :

  ∀ (i,j : 0 <= i <n and 0 <= j <n : i <j ==> a[i] <= a [j])

nếu R = true , ta có thể bỏ qua : ∀ (d :: 0 = d*0)

b) Lượng từ tồn tại.

Để khẳng định có một phần tử của tập hợp có tính chất P ta dùng lượng từ tồn tại ∃ (đọc là: " có một " hoặc " tồn tại ").

Ví dụ : để khẳng định có gía tri x trong array a ta viết :

∃(i : 0 <= i < n : a [i] = x)

Biểu thức này được đọc như sau :

∃(i	tồn tại một (số nguyên) i
: 0<= i < n	sao cho i nằm giữa 0 và n-1

: a[i] = x thoả điều sau a[i] bằng x

Dạng chung là : ∃(danh sách biến : R : P)

Mệnh đề tồn tại chỉ đúng khi có một phần tử trong miền xác định bởi R thoả P.

khi R = true thì ta có thể viết : ∃(danh sách biến :: P)

Ví dụ : cho hai array a và b

- Khẳng định :"trong array a không có thứ tự tăng"

  ∃(i : 0 <= i < n - 1 : a [i] >a [i+1])
- Khẳng định : "có ít nhất một phần tử của a lớn hơn mọi phần tử của b"

  ∃(i : 0 <= i <n : (j : 0 <= j < n : a[i] > b[j]))
- Khẳng định "n là chẵn" : ∃(m :: n = 2*m)

c) Một số tính chất:

- (i : R : P) ≡ (i :: R and P)
- (i : R : P) ≡ not (i :: R and not P)
- (i : R : P) ≡ not (i : R : not P)

d) Các biến tự do và bị buộc (free and band variables), phép thế(substitution)

Trong biểu thức Q(i: r(i) : p(i)) (ở đây ta xét Q là ∀ hay ∃) biến i được gọi là bị buộc (band) vào lượng từ Q .

Những xuất hiện của một biến i không bị buộc vào một lượng từ nào đó trong biểu thức R,được gọi là tự do (free) trong R.

Ví dụ trong biểu thức : ∃(d : p = q*d)

các biến p và q là tự do , còn biến d là bị buộc . Các biến bị buộc chỉ đóng vai trò "giữ chỗ" và có thể được đổi tên , nếu tên này không trùng với một biến tự do đã có. Vì vậy , biểu thức trên tương đương với :

∃(m :: p = q*m) nhưng hoàn toàn khác với : (p :: p = q*p)

Về nguyên tắc , một tên biến có thể vừa tự do và bị buộc trong cùng một biểu thức .

Ví dụ : Trong biểu thức ∀ (0<i) and (i : 0 <= i < n : a [i] = 0)

xuất hiện thứ nhất của i là tự do , còn xuất hiện còn lại là bị buộc.

Mặc dù ý nghĩa của biểu thức là rõ ràng nhưng nên tránh vì dễ gây nên lầm lẫn .

Xét một tân từ chứa biến tự do .

Ví dụ : is-divisor(q) ∃ (d :: p = q*d)

Ta có thể thay các xuất hiện tự do của một biến bằng một biểu thức để được một tân từ mới.

Ví dụ: thế 2*q cho q ta sẽ được tân từ is-divisor(2*q) mà dạng biểu thức của nó là : is-divisor(2*q) ∃ (d :: p = (2*q)*d)

Chú ý rằng trong ∃ (d :: p = q*d) biến p cũng tự do , nhưng vì ta không quan tâm đến phép thế cho p nên trong tân từ is-divisor(q) ta chỉ nêu q để giảm bớt đi các chi tiết không cần thiết trong diễn giải.

3. Tập hợp và tân từ.

Mỗi biến có thể lấy giá trị trong một tập hợp xác định . Tập trị mà một dãy các biến có thể nhận được là tích Descarters các tập trị của từng biến .

Ứng với một tân từ P(i), với i là (danh sách) biến tự do mà mỗi phép thế i bằng một hằng sẽ cho giá trị đúng hay sai , ta có một tập hợp tất cả các hàm mà phép thế i trong P cho giá trị đúng .

ký hiệu tập đó là : { i : P(i) }

Ví dụ : { i : i >= 0 } "tập các (số nguyên) i sao cho i không âm "

{ i,j : i < j } "tập các (số nguyên) i,j sao cho i nhỏ hơn j"

Ngược lại ứng với mỗi tập S , ta xây dựng tân từ đặc trưng cho S đó là:

P(i) = (i ∈ S)

Giữa các phép toán tập hợp và các phép toán logic có quan hệ chặt chẽ.

{ i : P(i) or Q(i) } ≡ { i : P(i) } U { i : Q(i) }

{ i : P(i) and Q(i) } ≡ { i : P(i) } ∩ { i : Q(i) }

Phần tử trung hoà của phép toán giao : tập vũ trụ (tích Descarters của các tập trị ứng với các biến trong danh sách biến) ứng với i chính là: { i : true }

Phần tử trung hoà của phếp toán hội là : { i : false }

4. Các lượng từ số học.

sử dụng ý tưởng của ∀ và ∃ ta đặt thêm các lượng từ số học để dơn giản hoá cách viết và dễ dàng áp dụng các phép biến đổi .

Mỗi lượng từ sau sẽ biểu thị một hàm số học :

- Lượng từ tổng S (sumation)

S(i: r(i): f(i)) chính là $\sum_i f(i)$ với i chạy trên tập hợp thoả r(i)

- Lượng từ tích P (product)

P(i: r(i): f(i)) chính là $\prod_i f(i)$ với i chạy trên tập hợp thoả r(i)

Qui ước :

S(i: false: f(i)) = 0

P(i: false: f(i)) = 1

- Lượng từ MAX và MIN

MAX (I: r(i): f(i)) là giá trị lớn nhất của f(i) trong các i thoả r(i).

MIN (I: r(i):f(i)) là giá trị nhỏ nhất của f(i) trong các i thoả r(i).

Qui ước :

MAX (i: false: f(i)) = - ∞

MIN (i: false: f(i)) = ∞

- Lượng từ N

N (i:r(i): P(i)) số giá trị i trong miền r(i) thoả P(i)

Tức là : N (i: r(i): P(i)) = S(i: r(i) and p(i): 1)

Mỗi lượng từ mà ta xét ngoại trừ N la` sự khái quát của các phép toán hai ngôi có tính giao hoán và kết hợp thành phép toán trên một tập bất kỳ.

Ví dụ :

S là khái quát của phép công (+), P là khái quát của phép nhân (*).